DCC205 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Aula 01

Carlos Bruno Oliveira Lopes
carlosbrunocb@gmail.com

Ementa

- Introdução à programação orientada a objetos (OO);
- Fundamentação da OO;
- Ampliação das definições de classe;
- Interação entre objetos;
- Agrupação de objetos;
- Comportamento sofisticado das classes;
- Análise do comportamento dos objetos;
- Design de classes;
- Herança;
- Técnicas adicionais de abstração.

Objetivo

Reconhecer e conceituar os elementos que compõem o paradigma orientado a objetos, analisar problemas, propor soluções e escrever programas numa linguagem orientada a objetos.

Bibliografia Básica

DEITEL, H. M. **Java : como programa**r. São Paulo : Pearson Prentice Hall, 2005.

SINTE, A. **Aprenda programação orientada a objetos em 21 dias**. São Paulo: Pearson Education do Brasil, 2002.

KEOGH JIM & GRANNINI MARIO. **OOP Desmistificado - Programação Orientada a Objetos**. Alta Books, 2005.

Bibliografia Complementar

BARNES, D. J. & KÖLLING, M. **Programação Orientada a Objetos com Java**. São Paulo : Pearson Prentice Hall, 2005.

BORATTI, ISAIAS CAMILO. **Programação orientada a objetos usando DELPHI**. Editora VISUAL BOOKS

MOTA , ALISSON ABREU. **Programação Orientada a Objetos com C++**. Relativa Editora, 2002.

AULA

01

“Surpreende-me ver que escritores antigos não tenham atribuído maior importância às leis da herança...”

Alexis de Tocqueville, *Democracy in America*, 1980

Programação Orientada a Objetos

Tipos de dados abstratos

- A abstração estende a noção de tipo sendo atrelado ao mecanismo de encapsulamento para definir novos tipos de dados.
- **Definição**: *Encapsulamento* é um mecanismo que permite que constantes logicamente relacionadas, tipos, variáveis, métodos, entre outros, sejam agrupados em uma nova entidade.
    - Ex.: Procedimentos, pacotes e classes.
- O encapsulamento é utilizado para limitar o escopo e a visibilidade dos valores de dados e das funções encapsuladas para esse tipo de dado recém-definido.
- (Tipo de dados abstrato)

Programação Orientada a Objetos

Tipos de dados abstratos

- Tipo STACK → definição de novo tipo

```
struct Node {
    int val;
    struct Node* next;
};
typedef struct Node* STACK;

int empty(STACK theStack);
STACK newstack( );
int pop(STACK* theStack);
void push(STACK* theStack, int newval);
int top(STACK theStack);
```

```
#include "stack.h"

struct Node {
    int val;
    struct Node* next;
};
typedef struct Node* STACK;

STACK theStack = NULL;

int empty( ) {
    return theStack == NULL;
}

int pop( ) {
    STACK temp;
    int result = theStack->val;
    temp = theStack;
    theStack = theStack->next;
    free(temp);
    return result;
}

void push(int newval) {
    STACK temp = (STACK)malloc(sizeof(struct Node));
    temp->val = newval;
    temp->next = theStack;
    theStack = temp;
}

int top( ) {
    return theStack->val;
}
```

Programação Orientada a Objetos

Modelo Objeto

- Evolução do conceito de modulação por funções;
- Mudança de modelo de abstração de estrutura de dados para objetos como atributos de encapsulamento, visibilidade e herança;
  - Linguagens orientada a objetos (programação orientada a objetos);
- A programação orientada a objeto:
  - Classe
  - Atributos
  - Métodos
  - Objetos

Programação Orientada a Objetos

Classes

Definição: *Classe* é uma declaração de tipo que encapsula constantes, variáveis e funções para manipulação dessas variáveis.

- **Atributos (variáveis de instâncias)**. Variáveis locais de uma classe que descrevem características do objeto modelado.
- **Métodos**. Funções locais de uma classe que descrevem ações que o objeto pode exercer.
  - **Construtores e destrutores**. Métodos (funções) especiais de uma classe usadas sua inicialização ou finalização.
- **Objetos**. São instâncias de classes.
- **Classe interna**. É uma classe implementada dentro de outra classe.
- **Cliente**. É qualquer outra classe ou método que declara ou usa um objeto de uma outra classe.

Programação Orientada a Objetos

Classes

```
class MyStack {
    class Node {
        Object val;
        Node next;
        Node(Object v, Node n) { val = v; next = n; }
    }
    Node theStack;

    MyStack( ) { theStack = null; }

    boolean empty( ) { return theStack == null; }

    Object pop( ) {
        Object result = theStack.val;
        theStack = theStack.next;
        return result;
    }

    Object top( ) { return theStack.val; }

    void push(Object v) {
        theStack = new Node(v, theStack);
    }
}
```

Programação Orientada a Objetos

Visibilidade e Ocultamento de informações

Em orientação a objetos é explícito o *ocultamento* e a *visibilidade* requerendo que todo método, toda variável de instância de classe (atributo), e classe tenha um nível particular de visibilidade, tais como:

- **`public`**. Visível a qualquer cliente e subclasse de classe.
- **`protected`**. Visível somente para uma subclasse da classe. (Em Java a todas as classes dentro do pacote)
- **`private`**. Visível à classe corrente.

Programação Orientada a Objetos

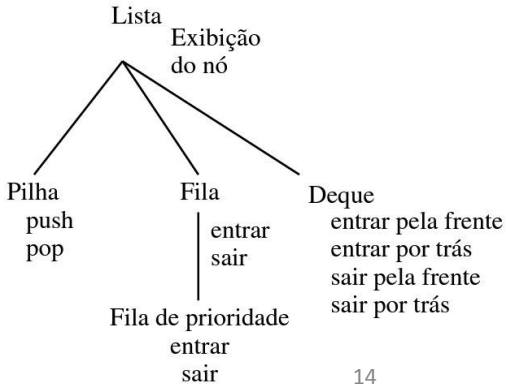
Visibilidade e Ocultamento de informações

```
public class MyStack {
    protected class Node {
        public Object val;
        public Node next;
        public Node(Object v, Node n) {
            val = v; next = n;
        }
    }
    private Node theStack;

    public MyStack( ) { theStack = null; }

    public boolean empty( ) { return theStack == null; }

    public Object top( ) { return theStack.val; }

    public Object pop( ) {
        Object result = theStack.val;
        theStack = theStack.next;
        return result;
    }

    public void push(Object v) {
        theStack = new Node(v, theStack);
    }
}
```

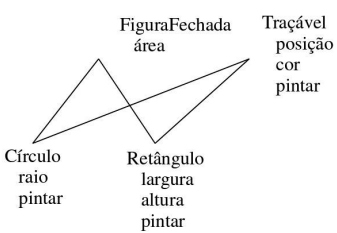
Programação Orientada a Objetos

Herança

- Reutilização de código;
- Forma uma hierarquia de classes para quais os códigos estão sendo herdados;
- Uma classe pode ser declarada com subclasse de outra classe (um especialização de uma classe generalizadora) chamada de ***classe-mãe*** ou ***superclasse***.
- Seu tipo de relação ***é uma*** ou ***é um***.



Programação Orientada a Objetos



Herança

- Há duas variedades de herança:
  - **Simples.** A hierarquia de classe forma uma árvore com sua raiz na classe mais geral. (Cada subclasse possui apenas uma superclasse).
  - ***múltipla***. Permite que uma classe seja subclasse de uma ou mais superclasses.

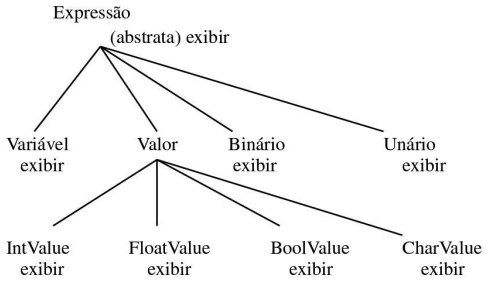
- **Definição**: uma *linguagem* é *orientada a objetos* se ela suporta um mecanismo de encapsulamento com ocultamento de informação para definir tipos de dados abstratos, métodos virtuais e herança.

Programação Orientada a Objetos

Herança


```
public class List extends Object {
    protected class Node {
        public Object val;
        public int priority;
        public Node prev, next;
        public Node (Object v, Node p) {
            val = v; prev = p; next = null; priority = 0; }
        public Node (Object v, Node p, Node n ) {
            val = v; prev = p; next = n; priority = 0; }
        public Node (Object v, int pr, Node p, Node n) {
            val = v; priority = pr; prev = p; next = n; }
    }
}
public class Stack extends List {
    private Node theStack;
    public MyStack( ) { theStack = null; }
    public Object pop( ) {
        Object result = theStack.val;
        theStack = theStack.next;
        return result;
    }
    public void push(Object v) {
        theStack = new Node(v, theStack);
    }
    ...
}
public class Queue extends List {
    protected Node front = null, rear = null ;
    public void enter(Object v) { ... }
    public Object leave( ) { ... }
    ...
}
public class PriorityQueue extends Queue {
    public void enter(Object v, int pr) { ... }
    public Object leave( ) { ... }
    ...
}
public class Deque extends List {
    public void enterfront(Object v) { ... }
    public void enterrear(Object v) { ... }
    public Object leavefront( ) { ... }
    public Object leaverear( ) { ... }
    ...
}
```

Programação Orientada a Objetos



Polimorfismo

- “Ter muitas formas”;
- **Definição**: Em Linguagens OO refere-se, à ligação tardia de uma chamada a uma ou várias diferentes implementações de um método em uma hierarquia de herança.

Modelo

- **Definição**: define uma família de classes parametrizadas por um ou mais tipos;

```
import java.util.ArrayList;
public class MyList<T> {
    private ArrayList<T> list;
    public MyList( ) { list = new ArrayList<T>( ); }
    public void add(T obj) { list.add(obj); }
    ...
}
```

Programação Orientada a Objetos

Classes Abstratas

- **Definição**: Uma *classe abstrata* é uma classe declarada como abstrata ou que tem um ou mais métodos abstratos;
- **Definição**: Um *método abstrato* é um método que não contém código além da assinatura.
- A subclasse que herda a classe abstrata que irá fornecer uma implementação para os métodos abstratos.

```
abstract class Expression { ... }
   class Variable extends Expression { ... }
   abstract class Value extends Expression { ... }
      class IntValue extends Value { ... }
      class BoolValue extends Value { ... }
      class FloatValue extends Value { ... }
      class CharValue extends Value { ... }
   class Binary extends Expression { ... }
   class Unary extends Expression { ... }
```

Programação Orientada a

```
public interface Map {
    public abstract boolean containsKey(Object key);
    public abstract boolean containsValue(Object value);
    public abstract boolean equals(Object o);
    public abstract Object get(Object key);
    public abstract Object remove(Object key);
    ...
}
```

Interface

- **Definição**: Uma interface encapsula uma coleção de constantes e assinaturas de métodos. Ela não pode incluir variáveis, construtores ou métodos não-abstratos.

```
public interface Comparable {
    public abstract int compareTo(Object obj);
}
```

```
public class Student implements Comparable {
    private String lastName;
    private String firstName;
    ...
    public int compareTo(Object obj) {
        Student other = (Student) obj;
        int comp = lastName.compareTo(other.lastName);
        if (comp != 0) return comp;
        return firstName.compareTo(other.firstName);
    }
}
```

Programação Orientada a Objetos

JAVA

- É uma linguagem mista que contém tipos de dados primitivos e objetos.
- Usa a semântica de cópia para tipos de dados primitivos e semântica de referência para objetos.
- Todos os métodos devem existir como parte da mesma classe.
- Usa coleta automática de lixo.
- É uma linguagem orientada a objetos estaticamente tipada, de herança simples.
- Ela tem suporte à:
    - Classes internas;
    - Visibilidade pública, protegida e privada para variáveis e métodos;
    - Classes abstratas;
    - Interfaces;
    - Modelos;

Programação Orientada a Objetos

JAVA

```
public abstract class Expression {
    public abstract Expression diff(Variable x);
}

class Value extends Expression {
    private int value;
    public Value(int v) { value = v; }
    public Expression diff(Variable x) {
        return new Value(0);
    }
}
class Variable extends Expression {
    private String id;
    static final private Value zero = new Value(0);
    static final private Value one = new Value(1);
    public Variable(String s) { id = s; }
    public Expression diff(Variable x) {
        return id.equals(x.id) ? one : zero;
    }
}
abstract class Binary extends Expression {
    protected Expression left, right;
    protected Binary(Expression u, Expression v) {
        left = u; right = v;
    }
}
class Add extends Binary {
    public Add(Expression u, Expression v) {
        super(u, v);
    }
    public Expression diff(Variable x) {
        return new Add(left.diff(x), right.diff(x));
    }
}
```