



UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BCC35A - Linguagens de Programação

Prof. Dr. Rodrigo Hübner

Aula 03: Tipo Abstrato de Dados e Suporte à Orientação a Objetos

Abstração, modularização e compilação separada

Introdução à abstração de dados

- Uma **abstração** é uma visão ou representação de uma entidade que inclui apenas os atributos mais significativos
 - String; Lista; Pilha; Button (UI); Sprite (jogo)
 - E todas as suas operações
- Ferramenta **contra a complexidade**
 - Simplifica desenvolvimento de software **encapsulando** a complexidade
 - Quando necessária uma string, lista, dicionário, conexão... Basta **saber como usar**

TAD: Tipo Abstrato de Dados

- Satisfaz duas condições:
 - A representação do tipo é escondida
 - As operações possíveis do TAD estão na definição do tipo
- As duas condições levam ao **encapsulamento de complexidade**
- Em linguagens com suporte a **Pacotes** ou **Classes**, as declarações de tipo estão em uma unidade estática (ex.: `.hpp` em `C++`), além de modificadores de acesso e criação de TADs.

Desenvolvimento X Linguagens OO

- **Suporte a OO:**
 - uma LP suportar OO não garante desenvolvimento OO
- Objetos possuem **atributos** e **métodos**
- Objetos possuem **responsabilidades**
- Comunicam-se com outros objetos via **mensagens**

Exemplo em C#

```
using System;

class Shape {
    protected string name;
    public Shape() : this("unnamed");
    public Shape(string name) { this.name = name; }
    public void display() {
        Console.WriteLine("Shape: {0}", this.name);
    }
}

class App {
    static void Main(string[] args) {
        Shape shape = new Shape("quadrado");
        shape.display();
        Console.WriteLine("\nPRESS [ENTER] TO QUIT");
        Console.ReadKey();
    }
}
```

Programação orientada a objetos

- Muitas LPs suportam programação OO
 - Estruturada + OO: `C++`, `Python` e `Ada`
 - Projetada p/ OO: `Java` e `C#`
 - Antigas: `Cobol` e `Lisp`
- Uma linguagem com suporte a OO **deve possuir**:
 - Definição de **TADs** p/ **encapsulamento de complexidade**
 - **Herança**
 - Vinculação dinâmica/virtual de métodos (**polimorfismo**)

Atributos de Objetos / Classes

- Encapsulamento: atributos não devem ser acessados diretamente
 - **Java**: *getters & setters*
 - **C#**: *properties*

Atributos de Objetos / Classes

Em **Java**...

```
class Shape {  
    protected string name;  
    public void setName(string name) {  
        this.name = name;  
    }  
    public void getName() {  
        return this.name;  
    }  
}
```


Atributos de Objetos / Classes

Em **C#**...

```
class Product {  
    protected string _model;  
    protected string _brand;  
  
    public string Model {  
        get { return _model; }  
        set { _model = value; }  
    }  
    // OU auto-property  
    public string Brand { get; set; }  
}
```

Instanciando objetos

- **Java** e **C#**: Instância de objetos em *Heap*
- **C#**: Programador escolhe *Stack* ou *Heap*
- **Python**, **Ruby**, **JavaScript**, **Dart**, ...: Sempre em *Heap*

```
int value;           // stack C++
int values[10];      // stack C++

int* value = new int(); // heap C++
*value = 100;
delete(p);
value = new int[10];
delete[] p;
```

Atributos de instância x classe

- Podemos especificar nas principais Ls OO dois formatos de atributos. No caso de `C#` pode ser: instância (*this*) e classe (*static*)

Ver e executar: `attr_meth_inst_class.cs`

Linguagens Orientadas a Objetos

- **Smalltalk**, **Ruby** e **Python3**: tudo é objeto; uniforme, porém baixo desempenho
- **C++**: tipos escalares tradicionais (+ array) e objetos; operações rápidas, porém sistema de tipos heterogêneo
- **C#**: tipos escalares tradicionais e objetos (+ array)
- Alocação de Tipos
 - Objetos: *Heap* (**Java**, **C#**, **C++**), *Stack* (**C++**)
 - Tipos escalares primitivos: *Stack* (**Java**, **C#**, **C++**), *Heap* (**C++**)

Orientação a Objetos em C++

```
class Shape {  
    protected:  
        string name;  
    public:  
        Shape(string name="sem nome") { this->name = name; }  
        virtual void display() {  
            cout << "Forma: " << this->name << endl;  
        }  
        virtual void something() = 0; // virtual pura -> abstract  
};
```

Orientação a Objetos em C++

```
class Rectangle: public Shape {  
    public:  
        Rectangle(string name="sem nome") : Shape(name) {}  
        virtual void display() { // "virtual" para indicar override  
            Shape::display();  
            cout << "Retângulo: " << this->name << endl;  
        }  
        virtual void something() {  
            cout << "Implementando o método abstrato" << endl;  
        }  
}
```

Testando *Shape* e *Rectangle* ...

```
void main() {  
  
    // alocação automática em stack  
    Shape shape1 = Rectangle("ret1"); // polimorfismo  
    shape1.display();  
    shape1.something();  
  
    // alocação manual em heap  
    Rectangle* shape2 = new Rectangle("ret2");  
    shape2->display();  
    shape2->something();  
    delete shape2;  
}
```

Classes aninhadas em C#

```
class Entity {  
    internal class Attr {  
        public Attr(string key, string value) {  
            this.Key = key;  
            this.Value = value;  
        }  
        public string Key { get; set; }  
        public string Value { get; set; }  
    }  
    private string name;  
    private Attr[] attrs;  
    public Entity(string name, Attr[] attrs) {  
        this.name = name;  
        this.attrs = attrs;  
    }  
}
```


Compilação separada

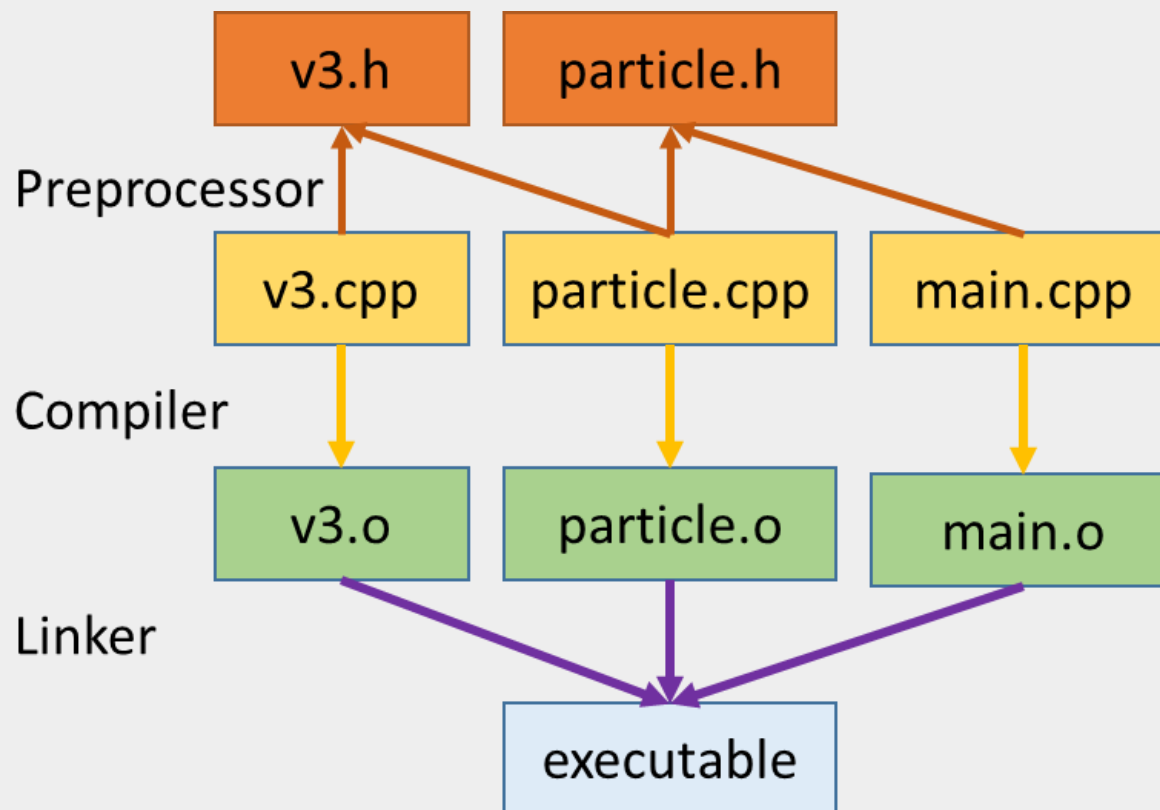
Não é específico das LPs com suporte a OO, porém a Orientação a objetos, abstrai partes de um programa em objetos específicos...

Compilação separada em C

```
#define DEBUG 1
int main() {
    #if DEBUG == 1
        printf("Descrição detalhada: ...\n");
    #elif DEBUG == 2
        printf("Descrição resumida: ...\n");
    #else
        printf("Nenhuma descrição\n");
    #endif
}
```

Compilando: `$ gcc D DEBUG=2 programa.c`

Compilação separada em C++



- Ver códigos de exemplo...

Próxima aula

- **Suporte à orientação a objetos:** polimorfismo; herança e composição; classes abstratas e abstratas puras (interfaces); templates.