

# *1. Encapsulamento*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



# Encapsulamento

- “Empacotamento” dos dados de uma classe (propriedades) para proteção de acesso externo.
- Manipulação indireta dos dados somente através de operações (métodos).
- Esconde detalhes de implementação irrelevantes.
- Permite alterar a estrutura interna dos dados mantendo a interface de operações.
- Relação com Tipos Abstratos de Dados (ADTs)
  - Ponto de vista do usuário: valores, operações e comportamento.
  - Ponto de vista do implementador: estruturas de dados concretas.

# Classe Retângulo

## ■ Propriedades

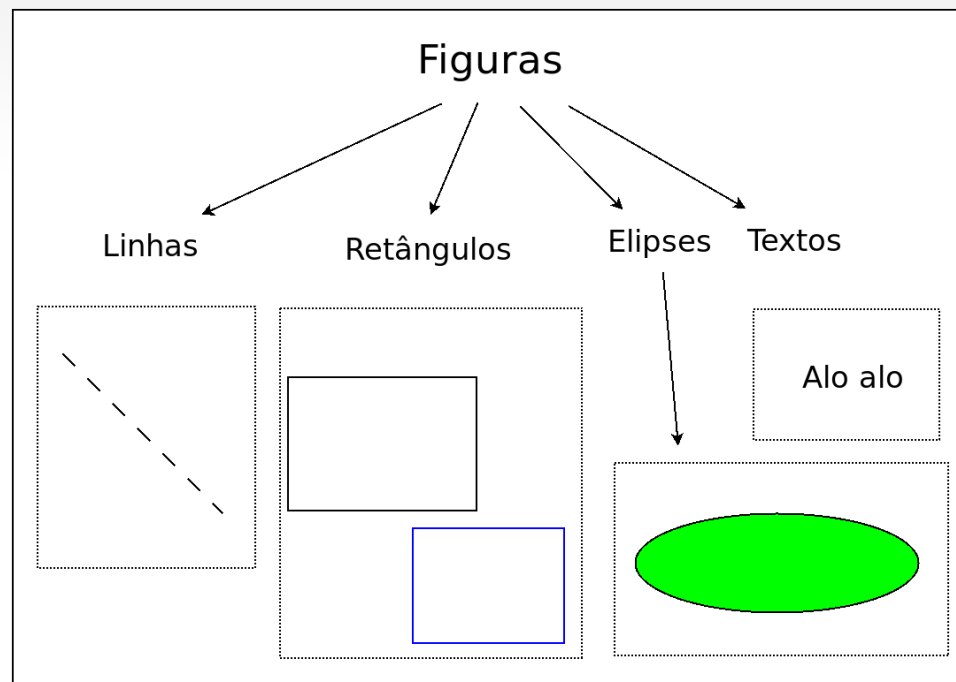
- $(x, y)$ : Posição
- $(w, h)$ : Tamanho

C

```
typedef struct {  
    int x, y;  
    int w, h;  
} Rect;
```

Java

```
class Rect {  
    int x, y;  
    int w, h;  
}
```



# Modificadores de Acesso

- Classes, métodos e propriedades
- *nenhum*: visível somente no pacote
- **private**: visível somente na classe
- **protected**: visível no pacote e subclasses
- **public**: visível por todos

## Evitar o modificador **public**

- a) Não usar nenhum modificador até que se faça necessário.
- b) Usar o modificador **private** até que se faça necessário.

# Exercício

- Passe um pente fino nas classes do seu projeto e use os modificadores mais restritos que forem possíveis para as suas propriedades.
  - **Não adicione novas operações.**
- 1. Enumere as modificações feitas.
- 2. Identifique um caso de uso para cada modificador (*public*, *protected*, *private*, *nenhum*) e explique a escolha.

# *1. Encapsulamento*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



## *2. Tipos Abstratos de Dados*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

[francisco@ime.uerj.br](mailto:francisco@ime.uerj.br)



# O Conceito de Abstração

An **abstraction** is a view or representation of an entity that includes only the most significant attributes.

- Categorias de abstrações

- processos

```
sortInt(list, listLen)
```

- dados

```
float d;
```

- Uma “arma” contra complexidade

- facilita no gerenciamento de programas



# Tipo Abstrato de Dado

- Invólucro que inclui
  - representação interna do tipo de dado
  - implementação das operações sobre o tipo de dado
- Detalhes desnecessários são escondidos
  - “information hiding”
- Acesso (API)
  - tipo abstrato + construtores + operações

# Classe Retângulo

- Operações

- `drag(dx, dy)` : Arrasta o retângulo
- `paint(g)` : Desenha o retângulo

```
class Rect {  
    private int x, y;  
    private int w, h;  
    public void drag (int dx, int dy) {  
        this.x += dx;  
        this.y += dy;  
    }  
    public void paint (Graphics g) {  
        g.drawRect(this.x,this.y, this.w,this.h);  
    }  
}
```

# Comparação com C

## ■ Operações

- `drag(dx, dy)` : Arrasta o retângulo
- `print()` : Exibe o retângulo

```
// rect.h
```

```
typedef struct Rect Rect;
```

```
Rect* rect_new (void);
```

```
void rect_drag (Rect* this, int dx, int dy);
```

```
void rect_print (Rect* this);
```

```
// rect.c
```

```
#include "rect.h"
```

```
typedef struct Rect { ... };
```

```
Rect* rect_new (void) { ... }
```

```
void rect_drag (Rect* this, int dx, int dy) { ... }
```

```
void rect_print (Rect* this) { ... }
```

# Exercício

- Implemente o tipo Rect em C usando a técnica de tipos abstratos de dados:
  - `rect.h`
  - `rect.c`
  - `main.c` (vide 10-ADTs/)
- Compile e execute o programa da seguinte forma:
  - `gcc -c rect.c`
  - `gcc main.c rect.o`
  - `./a.out`

## *2. Tipos Abstratos de Dados*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

[francisco@ime.uerj.br](mailto:francisco@ime.uerj.br)



## *3. Getter & Setter*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



# Getter & Setter

- Métodos públicos de acesso a propriedades privadas

```
class Rect {  
    private int x, y;  
    public int getX () {  
        return this.x;  
    }  
    public int setX (int x) {  
        this.x = x;  
    }  
}
```

# Getter & Setter

- Um contrassenso (um “anti-pattern”) ?
  - Quebra de encapsulamento
    - <https://www.yegor256.com/2014/09/16/getters-and-setters-are-evil.html>
- Por quê não expor a propriedade como pública?
  - Restringir somente a leitura/escrita (read/write-only)
  - Restringir a certos limites
  - Computar o valor de outra forma
  - Validar a entrada
  - Servir como proxy para um banco de dados



# Exercícios

1. Enumere as propriedades públicas do seu projeto e justifique se o uso de métodos de acesso é vantajoso (*getter & setter*).
  - Justifique
  - Altere o programa de acordo
2. Leia a discussão na página a seguir e destaque um ponto a favor e um contra o uso de métodos de acesso
  - <https://stackoverflow.com/questions/1568091/why-use-getters-and-setters-accessors>

## *3. Getter & Setter*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



## *4. IStackable*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

francisco@ime.uerj.br



# Interfaces & Encapsulamento

- Interface exporta métodos públicos somente
- Interface naturalmente impõe encapsulamento
- Propriedade somente por *getter & setter*
- Implementação somente acessível pela interface
- **Objetivo: pilha com duas implementações**

# IStackable

- Duas implementações
  - LinkedStack
  - ArrayStack

```
interface IStackable {  
    int   size ();           // quantos elementos  
    void push (int v);      // empilha inteiro  
    int   pop  ();           // desempilha inteiro  
}
```

11-IStackable/

# IStackable

```
public static void main (String[] args) {  
    IStackable s1 = new LinkedStack();  
    s1.push(10);  
    s1.push(20);  
    int v1  = s1.pop();  
    int sz1 = s1.size();  
    System.out.format("v=%d, sz=%d\n", v1, sz1);  
  
    IStackable s2 = new ArrayStack();  
    s2.push(10);  
    s2.push(20);  
    int v2  = s2.pop();  
    int sz2 = s2.size();  
    System.out.format("v=%d, sz=%d\n", v2, sz2);  
}
```

# Exercício

1. Implementar IStackable de duas formas:
  - Lista encadeada - LinkedStack:
    - LinkedList: size, addFirst, removeFirst
  - Array - ArrayStack:
    - ArrayList: size, add, remove
  - 11-IStackable/
2. A classe LinkedStack deve estender ou conter LinkedList? Justifique.

```
class LinkedStack extends LinkedList
    implements IStackable
{
    ...
}
```

```
class LinkedStack implements IStackable
{
    LinkedList l = new LinkedList();
    ...
}
```

## *4. IStackable*

Linguagem de Programação II

<https://github.com/fsantanna-uerj/LP2/>

Francisco Sant'Anna

[francisco@ime.uerj.br](mailto:francisco@ime.uerj.br)

