

# Programação Modular

Fundamentos de P00:  
modularidade, classes e  
objetos

Material adaptado do prof João Caram

Prof. Dr Johnatan Oliveira

# Programação modular

2

- ▣ Técnica de projeto (*design*) de software.
- ▣ Decomposibilidade de software.
- ▣ Cada módulo é uma unidade independente que provê uma funcionalidade específica de um sistema.
- ▣ Módulos se comunicam.

# Programação modular

3

- Imprescindível para a construção de sistemas de software modernos:
  - Flexibilidade;
  - Reutilização;
  - Manutenção.



# Orientação por objetos

4

- Orientação por objetos abstrai o mundo real utilizando objetos que interagem entre si.
- Utiliza o princípio da decomposibilidade para desenvolver sistemas modulares
  - Quais são os componentes de uma gestão de estoque?
  - Quais são os componentes de uma reunião virtual?

# Programação orientada por objetos

5

- POO, segundo Alan Kay:
  - troca de mensagens;
  - proteção e retenção locais possibilitando ocultar o estado ou o processo;
  - associação tardia e dinâmica de tudo o que for possível.

# Orientação por objetos

6

- Análise Orientada para Objetos (OOA/AOO):
  - Examina os requisitos de um sistema de uma perspectiva de classes e objetos, usando o vocabulário do domínio do problema.

# Orientação por objetos

7

- ▣ Projeto Orientado por Objetos (OOD/DOO):
  - ▣ Projeto no qual o processo da decomposibilidade em objetos é utilizado e modelos físicos e lógicos de objetos são descritos.

# Orientação por objetos

8

- ▣ Programação Orientada por Objetos (OOP/POO):
  - ▣ Implementação de programas organizados como coleções de objetos cooperativos.



# Orientação por objetos

9

- Análise orientada por objetos



- Projeto orientado por objetos



- Programação orientada por objetos

PAREI

# Linguagens OO

10

- ▣ Java, C#, Python, Eiffel, Object Pascal.
  - ▣ Porém: tipos básicos.
- ▣ C++, Ada, Perl, PHP entre outras:
  - ▣ incluem o conceito de classes e objetos, mas não são consideradas LPs OO em um sentido mais rigoroso.

11

# CLASSES E OBJETOS



# Classe

12

- ▣ Descrição de um Tipo Abstrato de Dados (TAD).
- ▣ Constituída por atributos (dados/características) e métodos (ações/comportamento).

# Classe

13

- Um conjunto de entidades semelhantes compõem uma classe de objetos.
- Ex: Classe automóvel.
  - Características: placa, velocidade atual, km percorridos, combustível restante no tanque.
  - Ações: Alterar velocidade, atualizar km percorridos, verificar combustível no tanque, abastecer.

# Objetos

14

- Um objeto representa uma entidade referenciável (identificada) de uma classe.
  - Instância de uma classe.

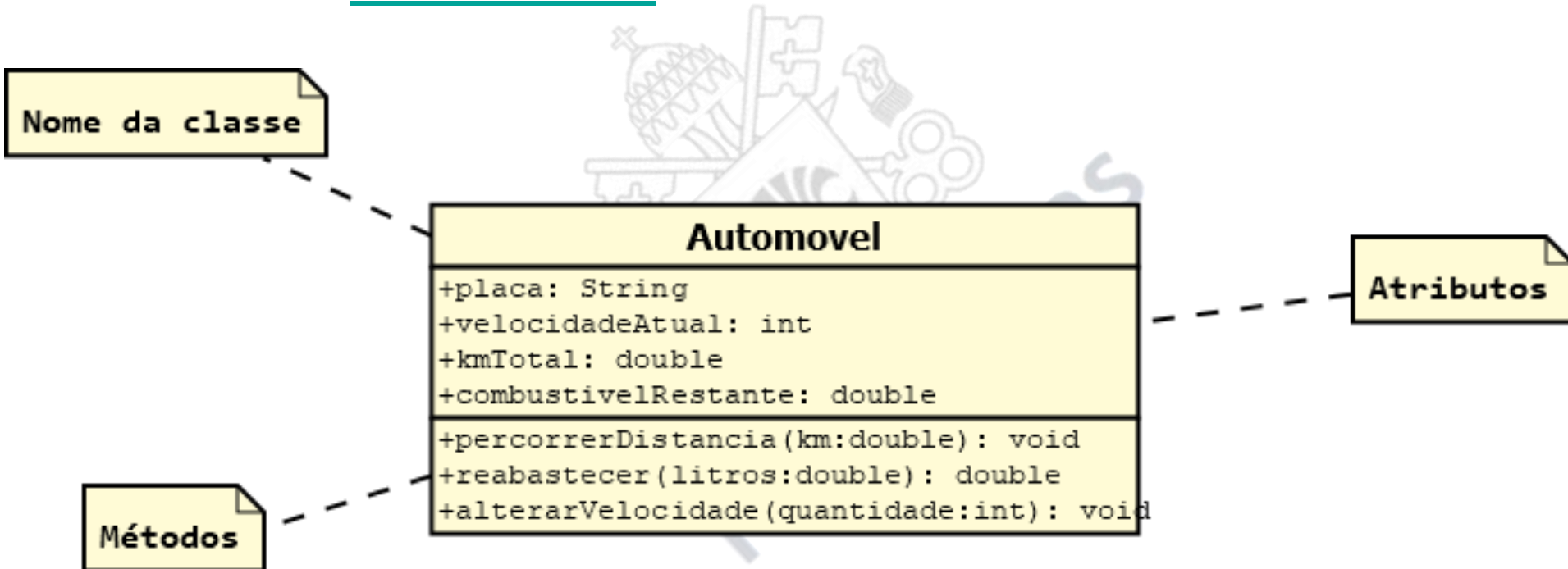
GAL0013  
71 km/h  
1908 km  
36,7 litros

PUC2023  
47 km/h  
25798 km  
11,2 litros

# Classes: representação UML

15

## ▣ Classe Automovel.



# Classes: exemplo/exercício

16

- Projete uma classe *Hora* para ser utilizada em relógios de sistemas diversos. A hora será representada até o nível de segundos.
- Um objeto *Hora* só pode armazenar estados válidos.
- Um objeto *Hora* pode receber incrementos de horas, minutos ou segundos.
- Um objeto *Hora* pode ser comparado com outro para verificação de qual valor está mais adiante.



# Pensando em Classes

17

- *Hora* com hora, minuto e segundo; validação, incremento e comparações.
- Atributos:
  - preco (double)
  - quant (int)
- Métodos:
  - temEstoque(): boolean
  - inicializarProduto(String, double, int): void

# Pensando em Classes

18

- *Hora* com hora, minuto e segundo; validação, incremento e comparações.
- Atributos:  
hora, minuto, segundo (int/byte)
- Métodos:  
ajustar(byte hora, byte min, byte seg): void  
validar(): boolean  
incrementar(byte quant, char posicao): void  
estahNaFrenteDe(Hora outra): boolean

# Pensando em Classes

19

- *Hora* com hora, minuto e segundo; validação, incremento e comparações.

## Hora

```
+hora: byte
```

```
+minuto: byte
```

```
+segundo: byte
```

```
+ajustar(hora:byte,min:byte,seg:byte): void
```

```
+validar(): boolean
```

```
+incrementar(quant:byte,posicao:char): void
```

```
+estahNaFrenteDe(outra:Hora): boolean
```

# Hora

```
class Hora {  
    byte hora;  
    byte minuto;  
    byte segundo;  
  
    boolean validar(){  
        if ( (hora>=0 && hora<=23) && (minuto>=0 && minuto<=59)  
            && (segundo>=0 && segundo<=59))  
            return true;  
        else{  
            hora = minuto = segundo = 0;  
            return false;  
        }  
    }  
}
```

Hora
+hora: byte +minuto: byte +segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): voi +validar(): boolean +incrementar(quant:byte,posicao:char): voi +estahNaFrenteDe(outra:Hora): boolean

# Hora

Hora

```
class Hora {  
    boolean estahNaFrenteDe(Hora outra){
```

+hora: byte

+minuto: byte

+segundo: byte

+ajustar(hora:byte,min:byte,seg:byte): void

+validar(): boolean

+incrementar(quant:byte,posicao:char): void

+estahNaFrenteDe(outra:Hora): boolean

```
}
```

# Hora

## Hora

+hora: byte

+minuto: byte

+segundo: byte

+ajustar(hora:byte,min:byte,seg:byte): void

+validar(): boolean

+incrementar(quant:byte,posicao:char): void

+estahNaFrenteDe(outra:Hora): boolean

```
class Hora {
    boolean estahNaFrenteDe(Hora outra){
        boolean resposta = false;
        if (hora > outra.hora)
            resposta = true;
        else if (hora == outra.hora)
            if(minuto > outra.minuto)
                resposta = true
            else if(minuto == outra.minuto)
                if(segundo > outra.segundo)
                    resposta = true;

        return resposta;
    }
}
```

# Hora

## Hora

+hora: byte

+minuto: byte

+segundo: byte

+ajustar(hora:byte,min:byte,seg:byte): void

+validar(): boolean

+incrementar(quant:byte,posicao:char): void

+estahNaFrenteDe(outra:Hora): boolean

```
class Hora {  
    boolean estahNaFrenteDe(Hora outra){  
        int esta;  
        esta = hora*3600 + minuto*60 + segundo;  
        int aquela;  
        aquela = outra.hora*3600 + outra.minuto*60 + outra.segundo;  
        return (esta > aquela);  
    }  
}
```

# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
        boolean naFrente = hora1.estahNaFrenteDe(hora2);  
        if(naFrente)  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
        naFrente = hora1.estahNaFrenteDe(hora2);  
    }  
}
```



# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
        boolean naFrente = hora1.estahNaFrenteDe(hora2);  
        if(naFrente)  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
        naFrente = hora1.estahNaFrenteDe(hora2);  
    }  
}
```

# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
        boolean naFrente = hora1.estahNaFrenteDe(hora2);  
        if(naFrente)  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
        naFrente = hora1.estahNaFrenteDe(hora2);  
    }  
}
```

# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
        boolean naFrente = hora1.estahNaFrenteDe(hora2);  
        if(naFrente)  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
        naFrente = hora1.estahNaFrenteDe(hora2);  
    }  
}
```

# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
        boolean naFrente = hora1.estaNaFrenteDe(hora2);  
        if(naFrente)  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
        naFrente = hora1.estaNaFrenteDe(hora2);  
    }  
}
```

# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
        boolean naFrente = hora1.estahNaFrenteDe(hora2);  
        if(naFrente)  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
        naFrente = hora1.estahNaFrenteDe(hora2);  
    }  
}
```

# Usando a classe Hora

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Hora hora1 = new Hora();  
        Hora hora2 = new Hora();  
        hora1.ajustar(20,24,45);  
        hora2.ajustar(20,24,42);  
  
        if(hora1.estaNaFrenteDe(hora2))  
            System.out.println("Hora1 na frente de Hora2");  
        else  
            System.out.println("Hora1 igual ou atrás de Hora2");  
        hora2.incrementar(1, 'm');  
    }  
}
```

31

# MODULARIDADE E POO



# Modularidade

32

*“Modularidade: mecanismo para aumentar a flexibilidade e compreensibilidade de um sistema, ao mesmo tempo em que permite a redução do seu tempo de desenvolvimento.”*

*(Parnas, David L. **On the Criteria To Be Used in Decomposing Systems into Modules.** Communications of the ACM, Vol. 15, No. 12, pp. 1053 – 1058, 1972. Tradução livre do autor.)*



# Modularidade

33

*“Modularidade: mecanismo para **aumentar** a **flexibilidade** e **compreensibilidade** de um sistema, ao mesmo tempo em que permite a **redução do seu tempo de desenvolvimento**.”*

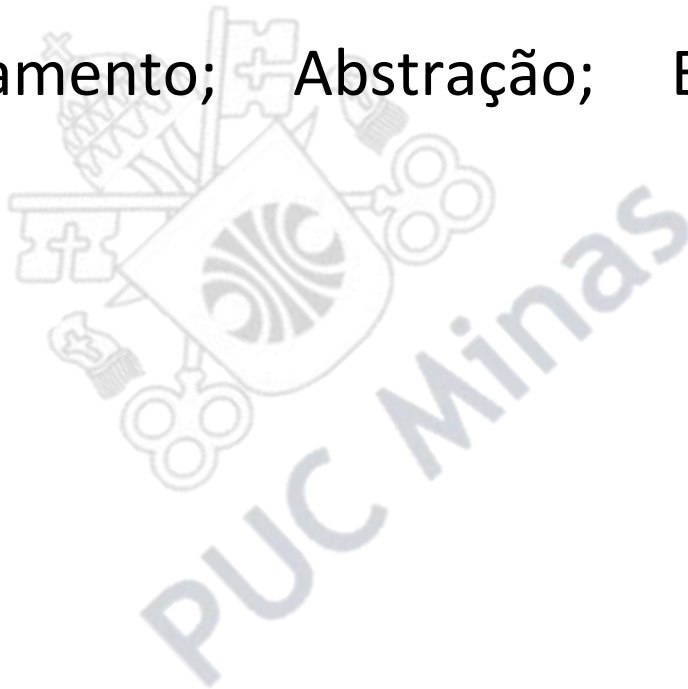
*(Parnas, David L. On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, Vol. 15, No. 12, pp. 1053 – 1058, 1972. Tradução livre do autor.)*

# Modularidade

34

## ▣ Quatro pilares em softwares modulares:

Coesão; Acoplamento; Abstração; Encapsulamento.



# Modularidade

35

- Quatro pilares em softwares modulares:  
Coesão; Acoplamento; Abstração; Encapsulamento.
- POO, segundo Alan Kay:
  - troca de mensagens;
  - proteção e retenção locais possibilitando ocultar o estado ou o processo;
  - associação tardia e dinâmica de tudo o que for possível.

# MODULARIDADE - COESÃO



# Coesão

37

- Coesão: Qualidade de uma coisa em que todas as partes estão ligadas umas às outras<sup>1</sup>
- Objetivo de um módulo em POO:
  - alta coesão interna;
  - dependência intramodular;
  - utilizar informações internas de forma coerente para resolver um problema específico.

1 - Dicionário Priberam da Língua Portuguesa, 2008-2022. Disponível em <<https://dicionario.priberam.org/coes%C3%A3o>>, acessado em Fev/2022

# Alta coesão

38

- Induz a independência funcional.
- Facilita a manutenção.
- Reduz efeitos colaterais e propagação de erros.

# Coesão e projeto OO

39

- POO, segundo Alan Kay:



- troca de *mensagens*;



- proteção e *retenção* locais possibilitando ocultar o estado ou o processo;

- associação tardia e dinâmica de tudo o que for possível.

# MODULARIDADE - ACOPLAMENTO





# Independência funcional

41

- ▣ Módulo: “*grupo de comandos com uma função bem definida e o mais independente possível em relação ao resto do algoritmo.*”
- ▣ A dependência pode ser medida pela quantidade de conexões entre os elementos de software.

# Acoplamento

42

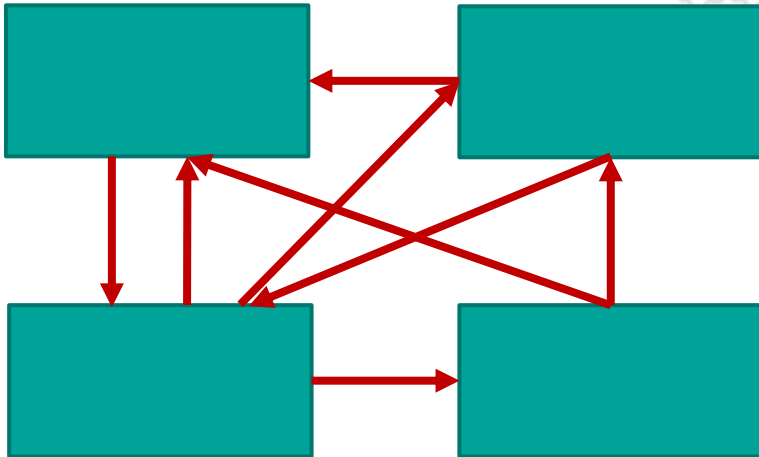
- Medida da interconexão entre elementos de software.



# Acoplamento

43

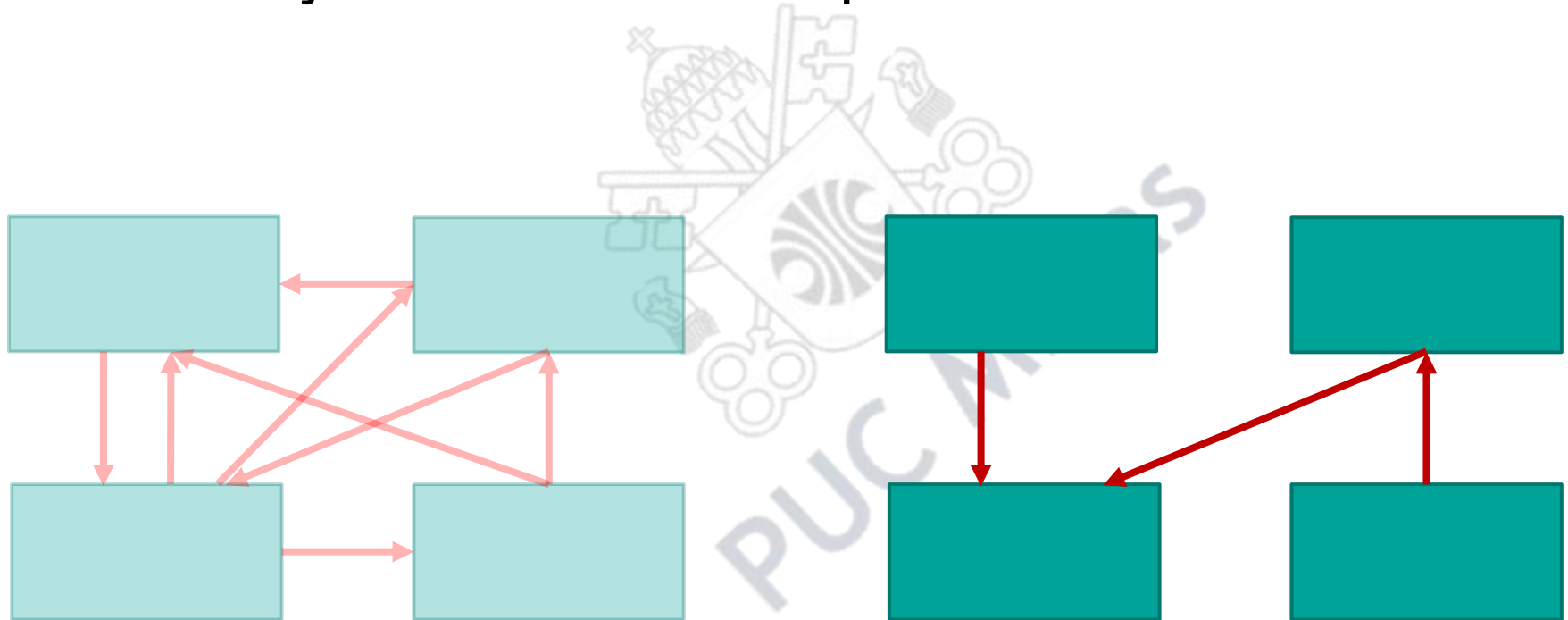
- "Espaguete": alto acoplamento.



# Acoplamento

44

- Situação ideal: baixo acoplamento.



# Baixo acoplamento: indicadores

45

- Tamanho:
  - Quantidade parâmetros e métodos públicos.
- Visibilidade:
  - Uso de parâmetros x Uso de variáveis globais.
- Flexibilidade:
  - Facilidade na chamada (abordaremos no futuro).

# Baixo acoplamento e projeto OO

46

- POO, segundo Alan Kay:

- troca de mensagens;



- proteção e retenção locais possibilitando ocultar o estado ou o processo;

- associação tardia e dinâmica de tudo o que for possível.

# MODULARIDADE - ABSTRAÇÃO



# Modelando classes

48

- Uma classe *Hora*.

Hora
+hora: byte +minuto: byte +segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +estahNaFrenteDe(outra:Hora): boolean



# Modelando classes

49

## ■ Quem define como é modelada a hora?

■ Int ou byte?

■ Centésimos, milésimos?

■ Só incrementa, não decrementa?

■ Mostra a diferença em segundos entre duas horas?

Hora
+hora: byte +minuto: byte +segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): voi +validar(): boolean +incrementar(quant:byte,posicao:char): voi +estahNaFrenteDe(outra:Hora): boolean

# Abstração

50

*“Capacidade de enxergar uma operação complexa de uma forma simplificada.”*

*“Visualização ou representação de uma entidade que inclui somente os atributos de importância em um contexto particular.” (Sebesta, 2000)*

# Abstração: representação

51

- Um módulo deve prover uma boa abstração da função pela qual é responsável:
  - Pessoa → cadastro de cidadãos (identidade);
  - Pessoa → cadastro acadêmico;
  - Pessoa → cadastro na *Epic Store*.
- Além disso...

# Abstração: operação simplificada

52

- Conceito da caixa-preta:
  - Entrada e saída bem conhecidas;
  - Detalhes ocultos.
- Ideia principal: não é necessário saber detalhes do funcionamento de um objeto para utilizá-lo.
  - Ex: como funciona o compartilhamento de tela em um aplicativo de reuniões *online*?

# Abstração: operação simplificada

53

- Caixa-preta e abstração:
  - Como o Spotify gerencia uma playlist?
  - Como a Netflix gerencia a lista de filmes para ver?
  - Como o Canvas gerencia a lista de disciplinas do aluno?

# Abstração e projeto OO

54

- POO, segundo Alan Kay:

- troca de mensagens;

-  □ proteção e retenção locais possibilitando ocultar o estado ou o processo;

-  □ associação tardia e dinâmica de tudo o que for possível.

55

# MODULARIDADE - ENCAPSULAMENTO



# Encapsulamento

56

- Encapsular: “incluir ou proteger em uma cápsula ou como em uma cápsula.”<sup>1</sup>
- Em POO:
  - Ocultar e proteger o estado;
  - Ocultar o processo.

1-Dicionário Michaelis Online. Disponível em < <https://michaelis.uol.com.br/busca?id=kQQD>>, acessado em Julho/2023



# Ex: um relógio e encapsulamento

```
class Hora {  
    public byte hora;  
    public byte minuto;  
    public byte segundo;  
    ...  
}
```

```
Hora hora1 = new Hora();  
hora1.hora = 22;  
hora1.minuto = 64;  
hora1.segundo = 93;  
hora1.incrementar(15, 's');
```

Hora
+hora: byte +minuto: byte +segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +estahNaFrenteDe(outra:Hora): boolean

# Ex: um relógio e encapsulamento

```
class Hora {  
    public byte hora;  
    public byte minuto;  
    public byte segundo;  
    ...  
}
```

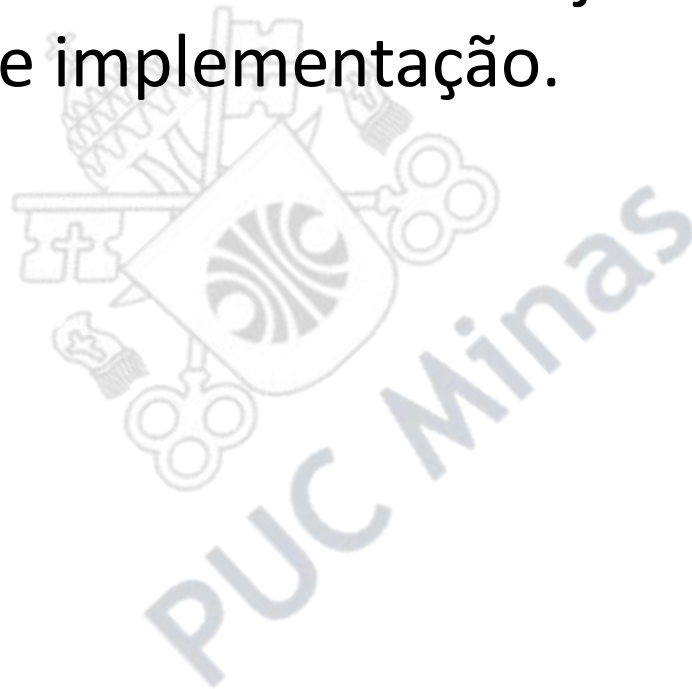
```
Hora hora1 = new Hora();  
hora1.hora = 22;  
hora1.minuto = 64;  
hora1.segundo = 93;  
hora1.incrementar(15, 's');
```

Hora
+hora: byte +minuto: byte +segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): voi +validar(): boolean +incrementar(quant:byte,posicao:char): voi +estahNaFrenteDe(outra:Hora): boolean

# Encapsulamento

59

- Separa aspectos visíveis de um objeto ou classe de seus detalhes de implementação.



# Encapsulamento

60

- Separa aspectos visíveis de um objeto ou classe de seus detalhes de implementação.
- Detalhes ocultos, interface exposta.
  - Interface: aquilo que o usuário do objeto vê;
  - Interface: maneira como o usuário utiliza o objeto.

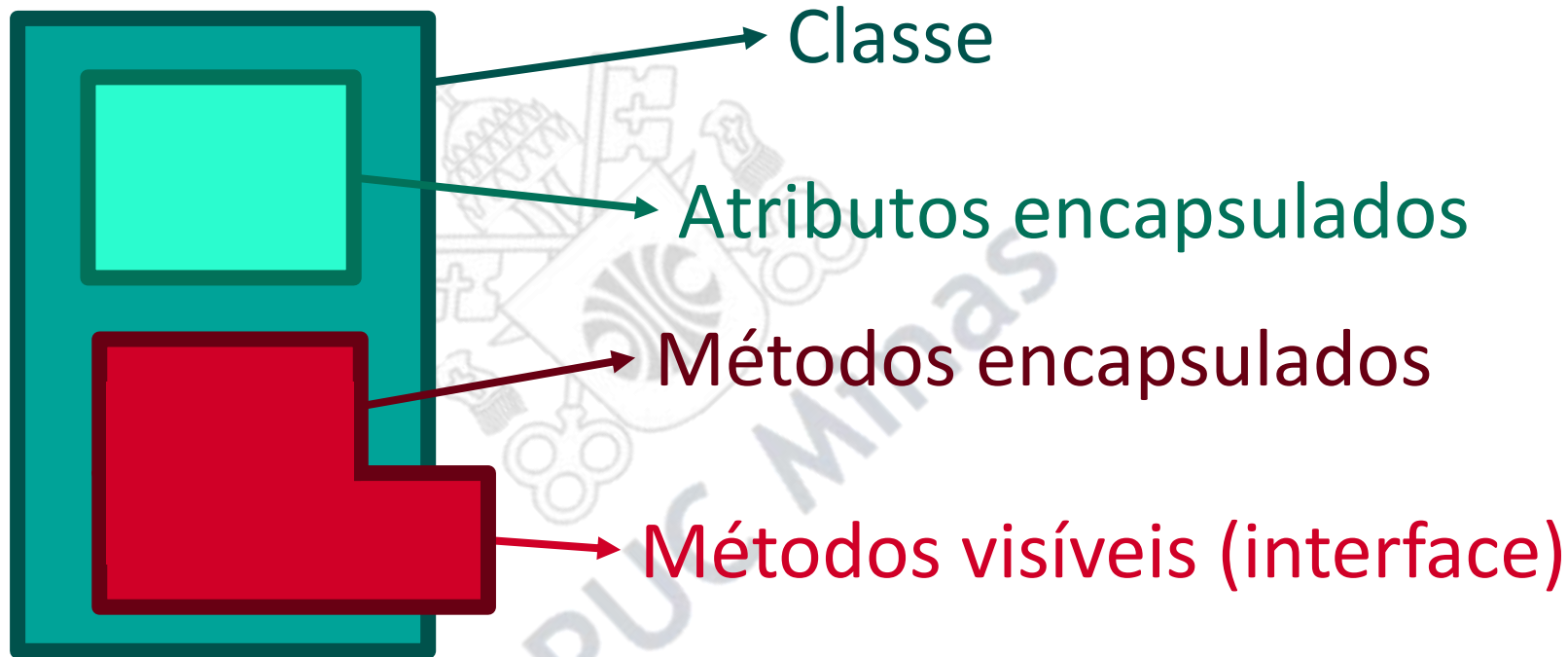
# Encapsulamento

61

- Permite alteração na implementação sem impactos em outros módulos do sistema;
- Favorece a abstração;
- Protege dados/atributos do acesso direto a partir de um código externo.

# Encapsulamento

62



# Encapsulamento e projeto OO

63

- POO, segundo Alan Kay:

- troca de mensagens;



- proteção e retenção locais possibilitando ocultar o estado ou o processo;

- associação tardia e dinâmica de tudo o que for possível.

# Níveis de acesso e modificadores

64



- Os níveis de acesso típico em linguagens OO são:
  - Público;
  - Privado;
  - Protegido.





# Níveis de acesso

65

-  Público (acesso irrestrito):
  - Classe, atributos ou métodos visíveis em qualquer parte.
-  Privado (acesso totalmente restrito):
  - Classe, atributos ou métodos visíveis somente para quem os declarou.

# Níveis de acesso

66

- # Protegido:
  - Classe, atributos ou métodos visíveis para quem o declarou e para módulos derivados deste.
  - Aplicação na *especialização com herança* (veremos mais adiante)

# Modificadores (Java)

67

- Diferentes linguagens usam palavras-chaves específicas, bem como podem implementar níveis particulares.
- Em Java, temos os modificadores:
  - Public, Private, Protected;
  - *Default (package).*

# Níveis de acesso (Java)

68

## ■ *Default (package):*

- Não é necessário declarar no Java;
- Classe, atributos e métodos visíveis para quem todos que fazem parte do mesmo pacote Java.
  - *Não há visibilidade para subclasses de outros pacotes.*

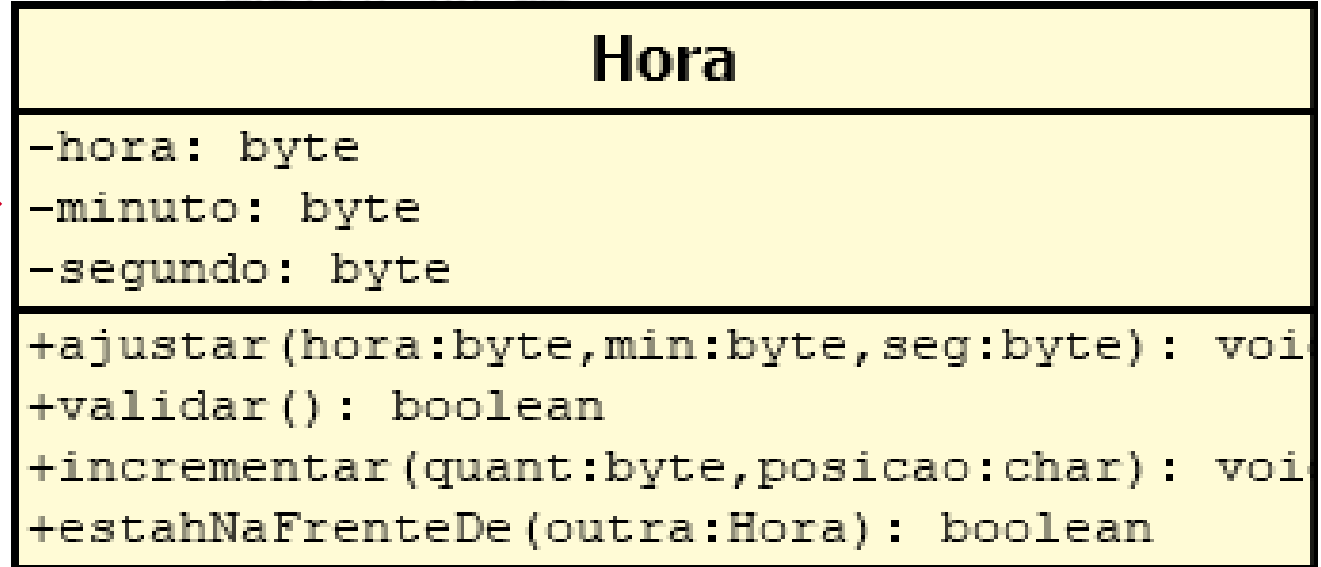
# Níveis de acesso (Java)

69

Modificador	Classe	Pacote	Subclasse	Mundo
<b>public</b>	sim	sim	sim	sim
<b>protected</b>	sim	sim	sim	não
<b>default</b>	sim	sim	não	não
<b>private</b>	sim	não	não	não

# Ex: um relógio e encapsulamento

```
class Hora {  
    private byte hora;  
    private byte minuto;  
    private byte segundo;  
    ...  
}
```



# Ex: um relógio e encapsulamento

```
class Hora {  
    private byte hora;  
    private byte minuto;  
    private byte segundo;  
    ...  
    public void ajustar(.....){  
        ...  
    }  
    public void incrementar(.....){  
        ...  
    }  
}
```

Hora
-hora: byte
-minuto: byte
-segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void
+validar(): boolean
+incrementar(quant:byte,posicao:char): void
+estahNaFrenteDe(outra:Hora): boolean

# Ex: um relógio e encapsulamento

```
class Hora {  
    private byte hora;  
    private byte minuto;  
    private byte segundo;  
    ...  
}  
  
Hora hora1 = new Hora();  
hora1.ajustar(22,64,93);  
hora1.incrementar(15, 's');
```

Hora
-hora: byte -minuto: byte -segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +estahNaFrenteDe(outra:Hora): boolean



# Ex: um relógio e encapsulamento

```
class Hora {  
    private byte hora;  
    private byte minuto;  
    private byte segundo;  
    ...  
}  
Hora hora1 = new Hora();  
hora1.ajustar(22,64,93);  
hora1.incrementar(15, 's');
```

Hora
-hora: byte -minuto: byte -segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +estahNaFrenteDe(outra:Hora): boolean

# Ex: um relógio e encapsulamento

```
class Hora {  
    private byte hora;  
    private byte minuto;  
    private byte segundo;  
    ...  
}
```

```
Hora hora1 = new Hora();  
hora1.ajustar(22,64,93);  
hora1.incrementar(15, 's');
```

Hora
-hora: byte
-minuto: byte
-segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void
+validar(): boolean
+incrementar(quant:byte,posicao:char): void
+estahNaFrenteDe(outra:Hora): boolean

# Ex: um relógio e encapsulamento

```
class Hora {  
    private byte hora;  
    private byte minuto;  
    private byte segundo;  
    ...  
}  
  
Hora hora1 = new Hora();  
hora1.ajustar(22,64,93);  
hora1.incrementar(15, 's');
```

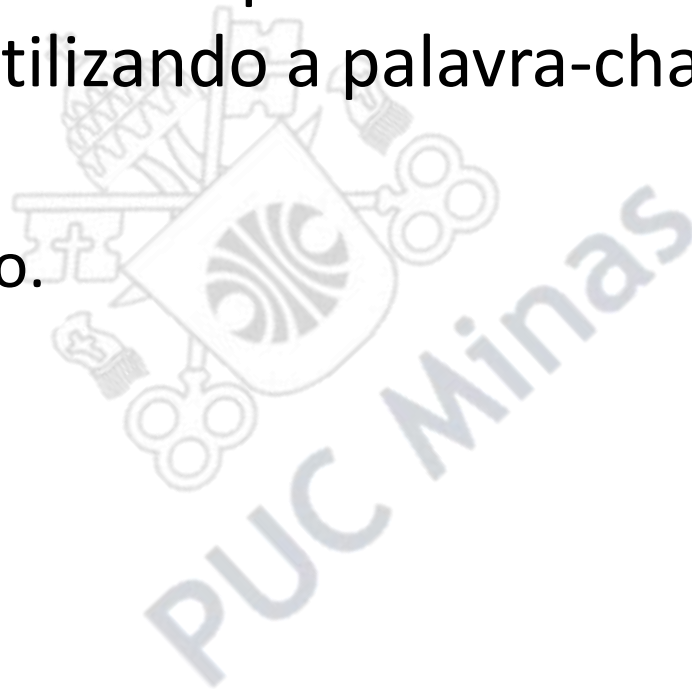
Hora
-hora: byte -minuto: byte -segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +estahNaFrenteDe(outra:Hora): boolean

Qual horário o  
objeto está  
armazenando  
agora?

# Encapsulamento

76

- Internamente, a classe pode se referir a seus componentes utilizando a palavra-chave this.
  - Clareza;
  - Desambiguação.



# Hora

---

```
class Hora {  
    public boolean validar(){  
        if ((this.hora>=0 && this.hora<=23) &&  
            (this.minuto>=0 && this.minuto<=59) &&  
            (this.segundo>=0 && this.segundo<=59))  
            return true;  
        else{  
            this.hora = this.minuto = this.segundo = 0;  
            return false;  
        }  
    }  
}
```

# Encapsulamento

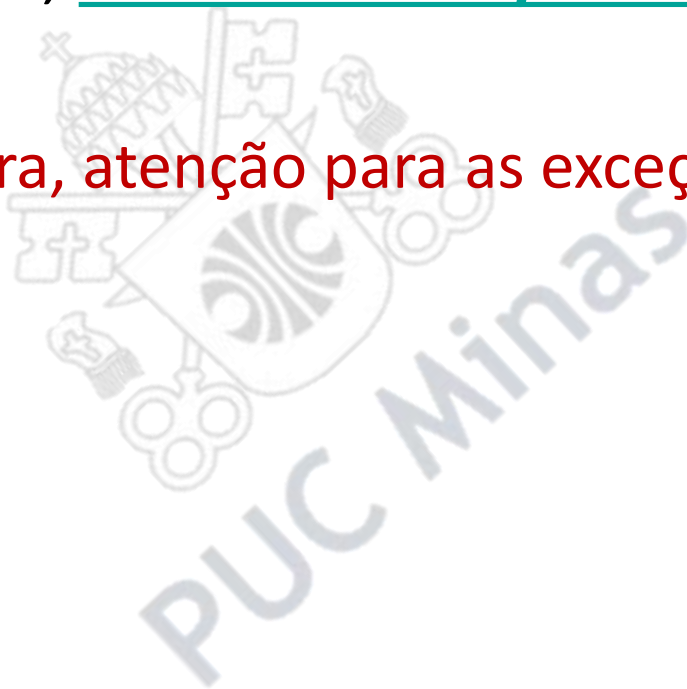
78

- ▣ Os atributos de uma classe só devem ser acessados e modificados por meio de seus métodos.
- ▣ Um método de uma classe só deve existir se ele tiver relação com os atributos da classe.

# Encapsulamento

79

- ▣ De maneira geral, *atributos são privados* e *métodos são públicos*;
  - ▣ Como toda regra, atenção para as exceções;



# Hora

---

```
class Hora {  
    public boolean validar(){  
        if ((this.hora>=0 && this.hora<=23) &&  
            (this.minuto>=0 && this.minuto<=59) &&  
            (this.segundo>=0 && this.segundo<=59))  
            return true;  
        else{  
            this.hora = this.minuto = this.segundo = 0;  
            return false;  
        }  
    }  
}
```



# Hora

---

```
class Hora {  
    private boolean horaValida(){...}  
    private boolean minutoValido(){...}  
    private boolean segundoValido(){...}  
  
    public boolean validar(){  
        if (horaValida() && minutoValido() && segundoValido())  
            return true;  
        else{  
            this.hora = this.minuto = this.segundo = 0;  
            return false;  
        }  
    }  
}
```

# Métodos de acesso

82

- Métodos *get*: dão acesso ao valor de um atributo.
- Métodos *set*: realizam a atribuição segura de um valor a um atributo.
- Na recomendação original, em Java os *getters* e *setters* usam o prefixo e o nome do atributo encapsulado.

# Métodos de acesso

83

■ Ex:

Hora
-hora: byte -minuto: byte -segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +estahNaFrenteDe(outra:Hora): boolean

int getHora();

void setMinutos(byte minutos);

# Métodos de acesso

84

■ Ex:

Hora
-hora: byte -minuto: byte -segundo: byte
+ajustar(hora:byte,min:byte,seg:byte): void +validar(): boolean +incrementar(quant:byte,posicao:char): void +tahNaFrenteDe(outra:Hora): boolean

int getHora()

void

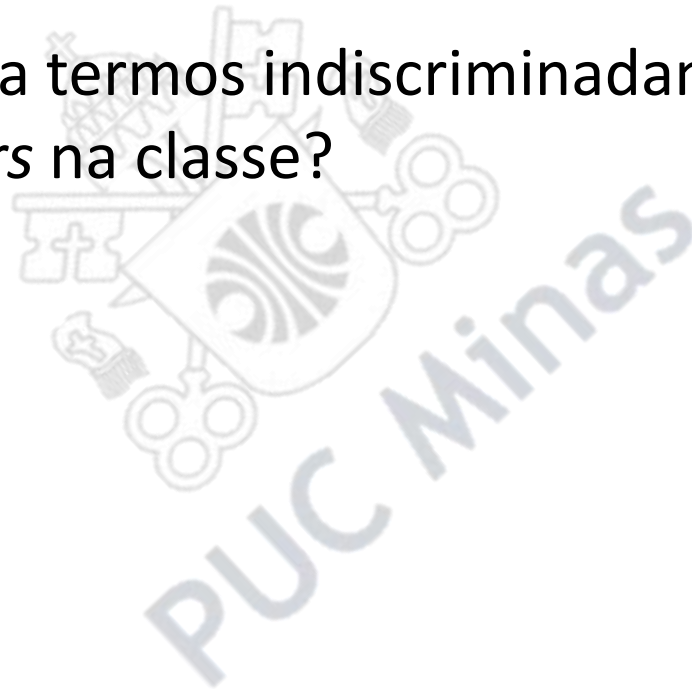
(byte minutos);

SERÁ?!?!?!!

# Getters and setters

85

- ▣ Debate:
  - ▣ É uma boa idéia termos indiscriminadamente métodos *getters* e *setters* na classe?



# Getters and setters

86

## ▣ Why getter and setter methods are evil.

<https://www.infoworld.com/article/2073723/why-getter-and-setter-methods-are-evil.html>

## ▣ Getters/Setters. Evil. Period.

<https://www.yegor256.com/2014/09/16/getters-and-setters-are-evil.html>