Desenvolvimento de Sistemas Orientados a Objetos I

Princípios e conceitos da Orientação a Objetos

Jean Carlo Rossa Hauck, Dr.

jean.hauck@ufsc.br

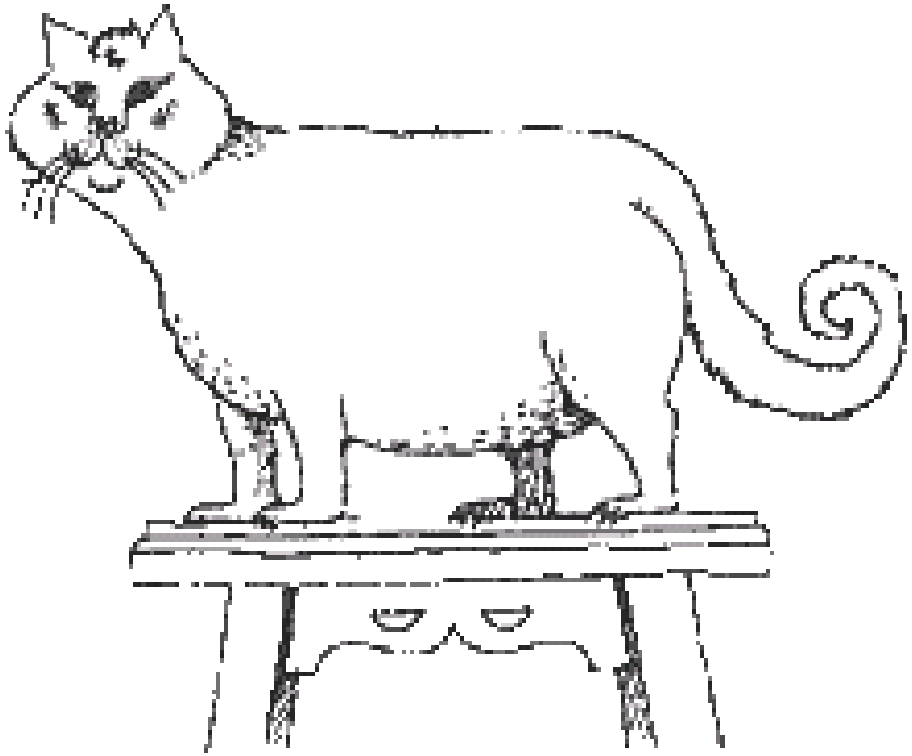
http://www.inf.ufsc.br/~jeanhauck

Conteúdo Programático

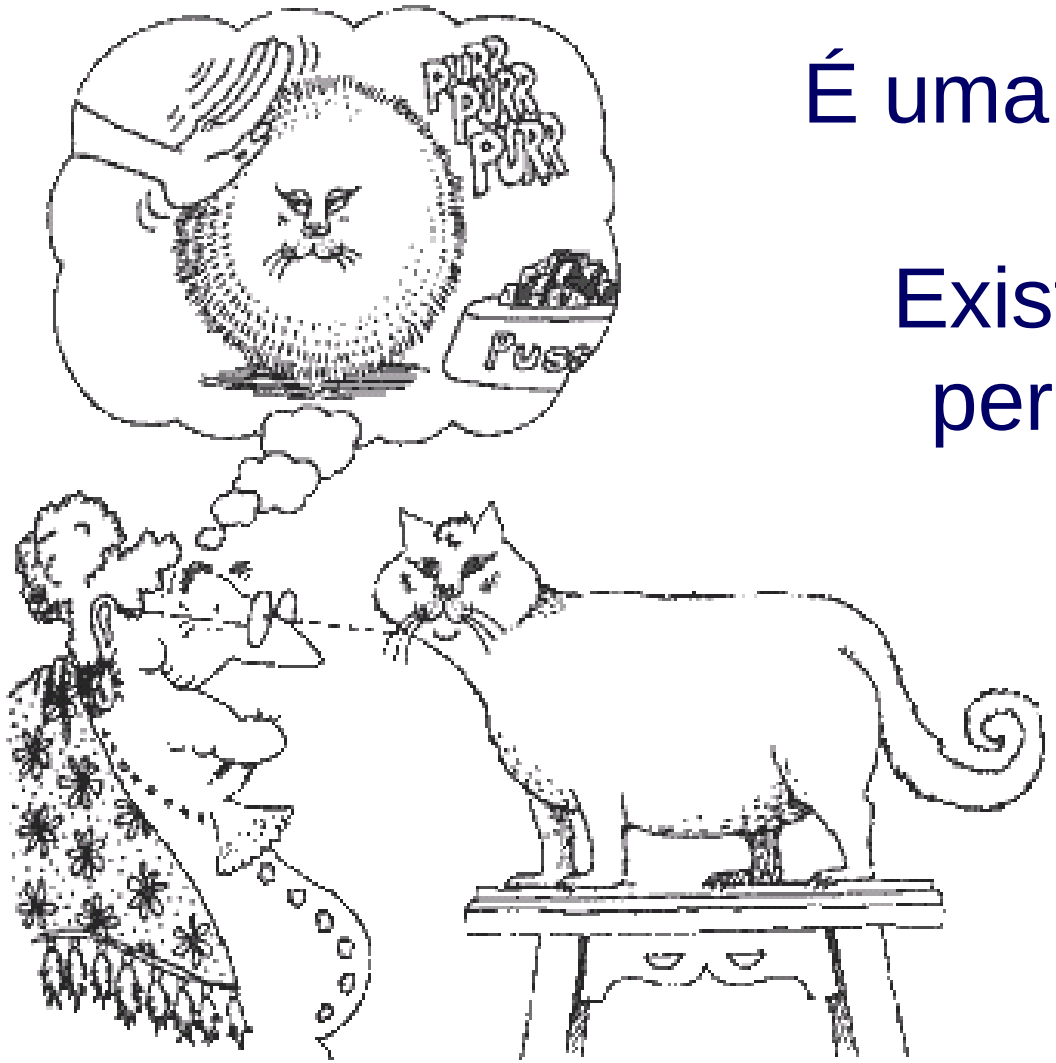
- Introdução
  - Introdução ao desenvolvimento de sistemas reusáveis de software
- Conceitos e mecanismos da programação orientada a objetos
  - Objetos e classes

Abstração

- O que você está vendo aqui?
- Qual a sua percepção?



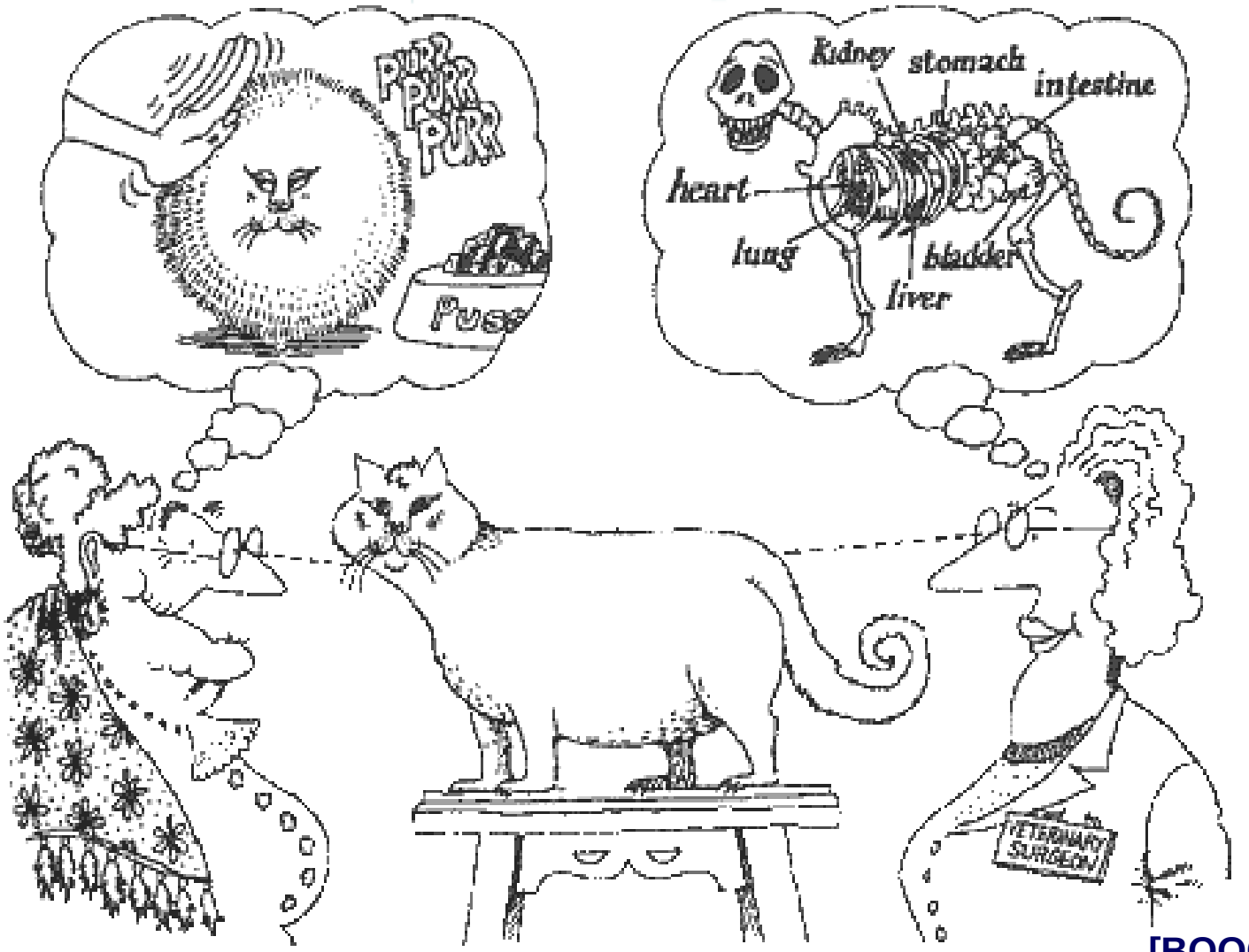
Abstração



É uma visão única?

Existem outras percepções?

Abstração

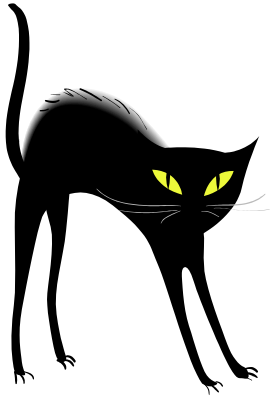


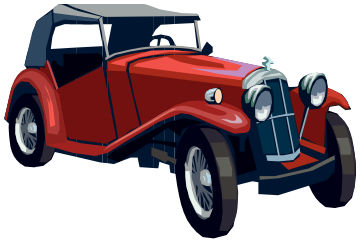
[BOOCH, 1994]

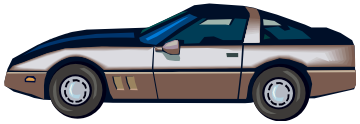
Abstração

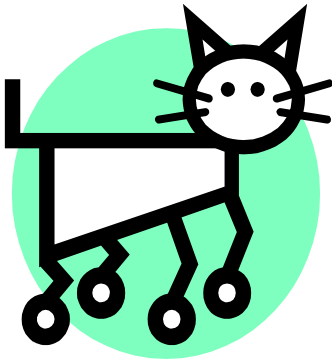
- Identificar aspectos importantes de um fenômeno e ignorar detalhes
- Técnica para lidar com a complexidade
- A seleção de quais aspectos são importantes depende do observador e do problema observado
- Criação de modelos: protótipos, equações, etc

Objetos



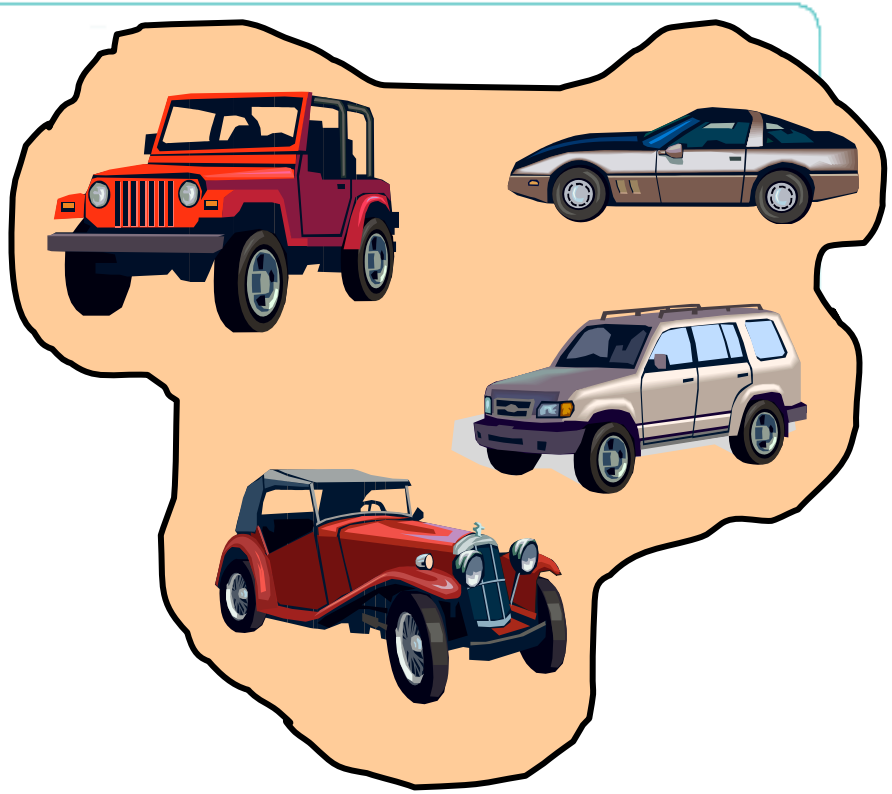






Classes dos Objetos

Gato



Carro

Atributos

- Número de pás: 3
- Tipo de pá: MADEIRA
- Número de velocidades: 3
- Cor: MOGNO
- Tem exaustor: SIM

Atributos

- Número de pás: 2
- Tipo de pá: PLÁSTICO
- Número de velocidades: 4
- Cor: VERDE
- Tem exaustor: SIM

Atuando sobre um objeto

- Altera o **número de pás**
- Qual o **número de pás**?
- Altera a **cor**
- Qual a **cor**?
- ...

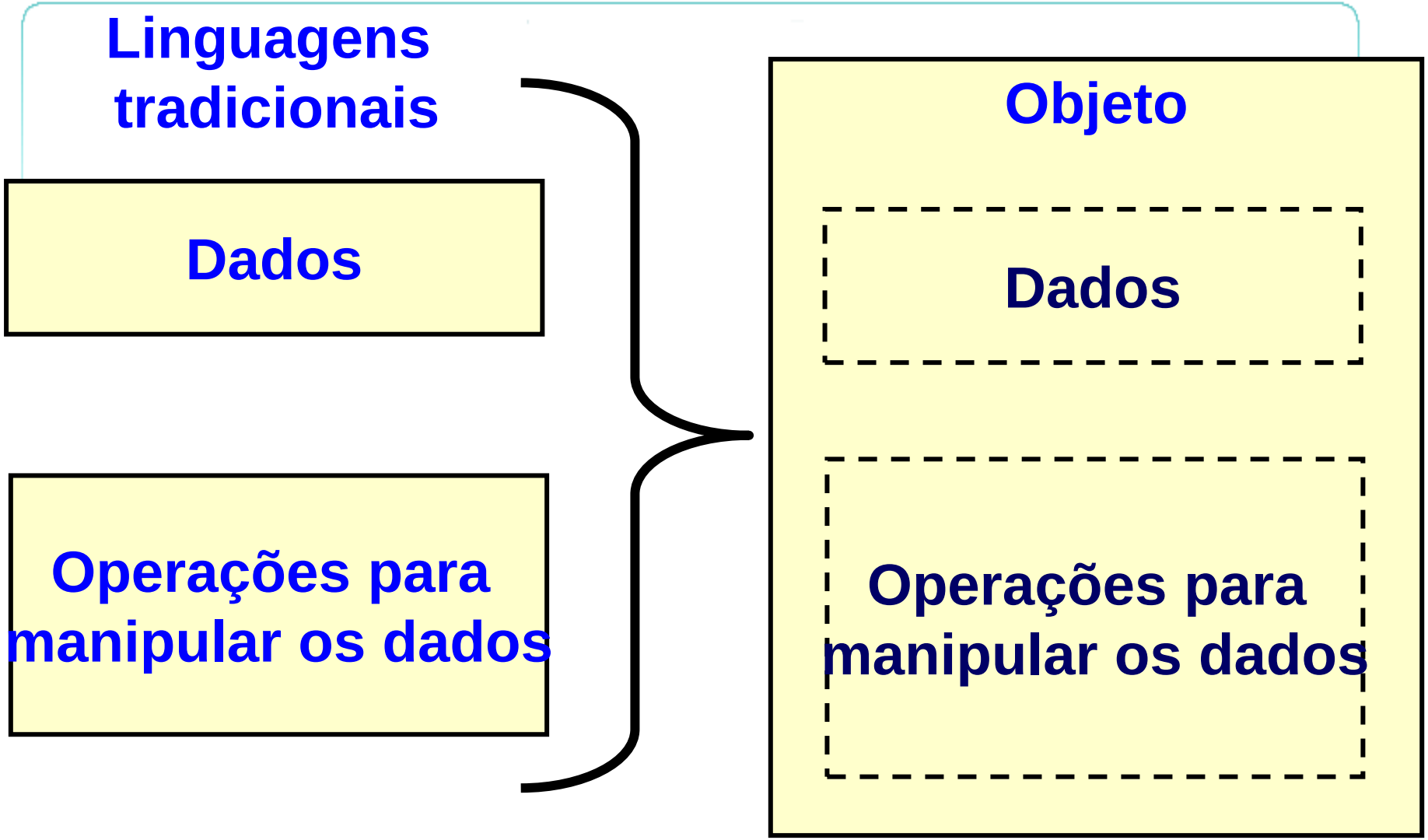
Outros atributos do objeto

- Status do ventilador: Desligado
- Status da luz: Desligado
- Velocidade atual: 0
- ...

Outras ações sobre um objeto

- Liga
- Desliga
- Aumenta velocidade
- Reduz velocidade
- Acende luz
- Apaga luz
- ...

Entendendo um objeto



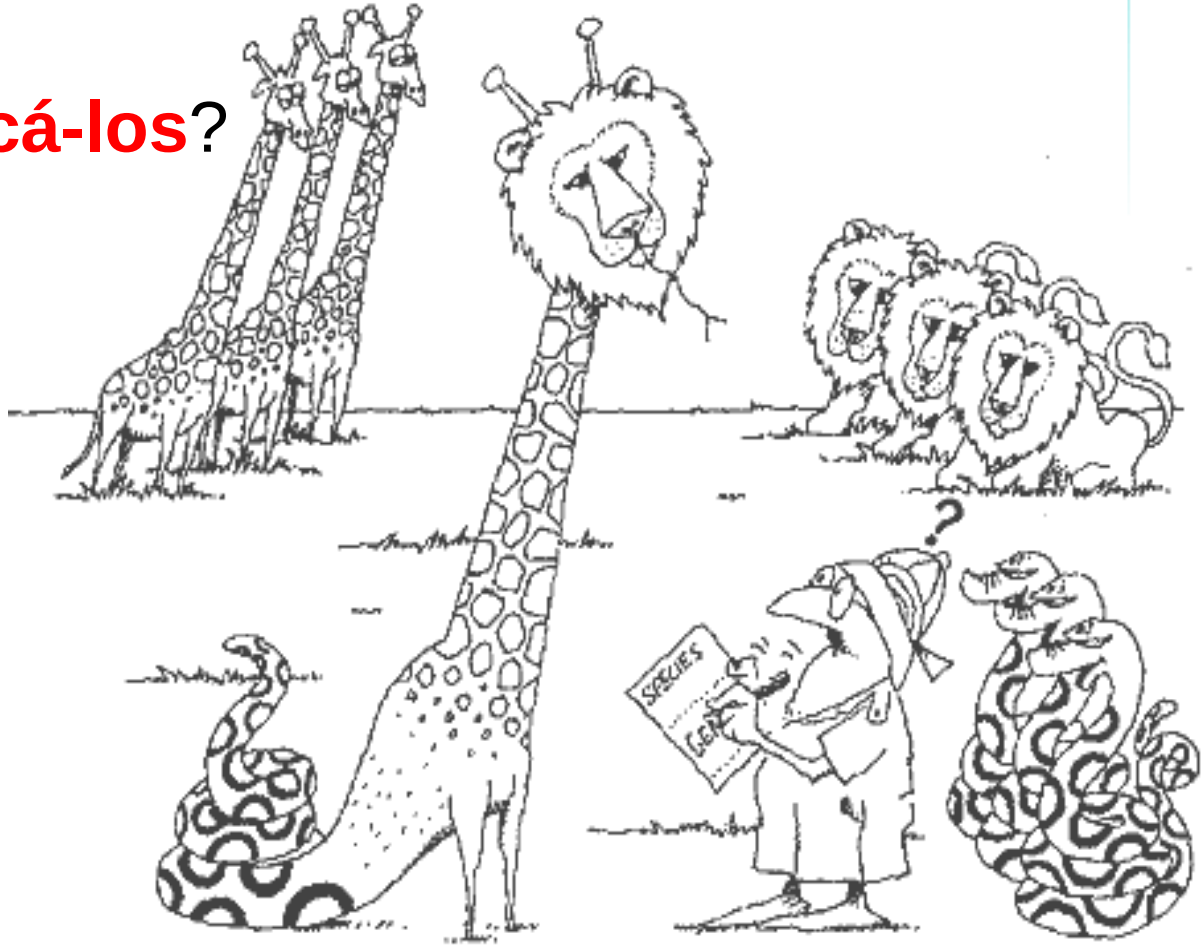
Conceito de objeto

- Uma representação computacional de **algo** que pode ser utilizado para **executar uma tarefa**
- Um objeto é um conceito, abstração, ou algo com **limites bem definidos** e **significado bem conhecido** dentro de uma aplicação
- Uma **coleção de dados** e **operações** para **manipular estes dados**, representando uma entidade lógica no sistema

Classificação dos Objetos

Classificação dos objetos

- Existem muitos objetos com características e comportamentos **similares**
- Como **classificá-los**?



[BOOCH, 1994]

Classificação dos objetos

- Existem muitos objetos com características e comportamentos **similares**
- C

Abstração!!

[BOOCH, 1994]

Diferentes abstrações?

- O que você está vendo?

[BOOCH, 1994]

Conceito de classe

- Uma classe é a **definição de um grupo de objetos** com propriedades (atributos), comportamento (operações) e semântica comuns
  - **Exemplos**: Ventilador, Pessoa, Carro, ContaCorrente, etc
- Uma classe pode ser entendida como um **molde para gerar objetos com a mesma estrutura** (atributos/operações/relacionamentos)
- Classes são implementadas; objetos são utilizados
- Um objeto é uma **instância** de uma classe

Identificando e nomeando classes

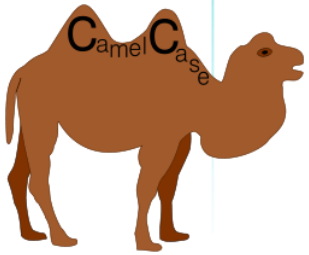
- Uma classe deveria capturar uma e **somente uma abstração chave**
- Classes podem ser identificadas examinando-se os **substantivos** importantes do sistema a ser modelado
- Uma classe deveria ser um **substantivo singular** que melhor caracteriza a abstração: **Professor**, **Aluno**, **Turma**, ...
- Dificuldades na nomeação das classes podem indicar abstrações mal definidas
- Os **nomes** devem surgir diretamente do **domínio do problema**

Alguns cuidados

- Na identificação das classes, procure observar a **estrutura** destas classes → **objetos compartilham a mesma estrutura** (mesma classe), mas podem ter **estados diferentes**
- Usualmente, uma nova classe deve ter uma nova estrutura (atributos + operações)
- Por exemplo, quais as diferenças entre classes como **Cliente** e **Fornecedor**

Como nomear uma Classe

- Para nomear **Classes**, adote o estilo de escrita **UpperCamelCase**[1]
  - Quando o nome da classe for composto por mais de uma palavra, elas são concatenadas diretamente
  - Cada palavra é iniciada com uma **letra maiúscula**
  - Ex: ContaBancaria, Disciplina, Curso, FiguraGeometrica, ...



https://www.python.org/dev/peps/pep-0008/

Como nomear uma Classe

- Considerar a métrica de **Coesão**
  - Representa a relação existente entre as operações de uma determinada classe e de sua responsabilidade
  - Métrica **interna** a uma classe

- O objetivo é manter **ALTA COESÃO**
  - Quanto maior for a coesão, melhor aplicado foi o **Princípio da Responsabilidade Única (S**OLID**)**

Como nomear um Atributo

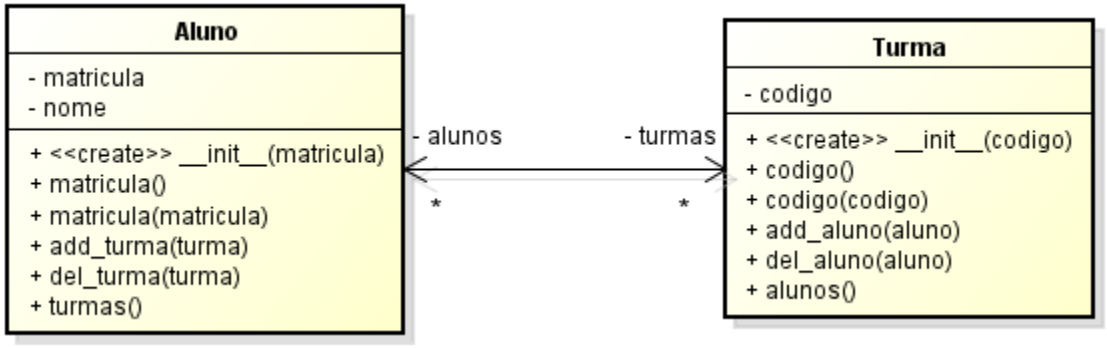
- Adote um estilo de escrita **snake_case** para

Atributos:

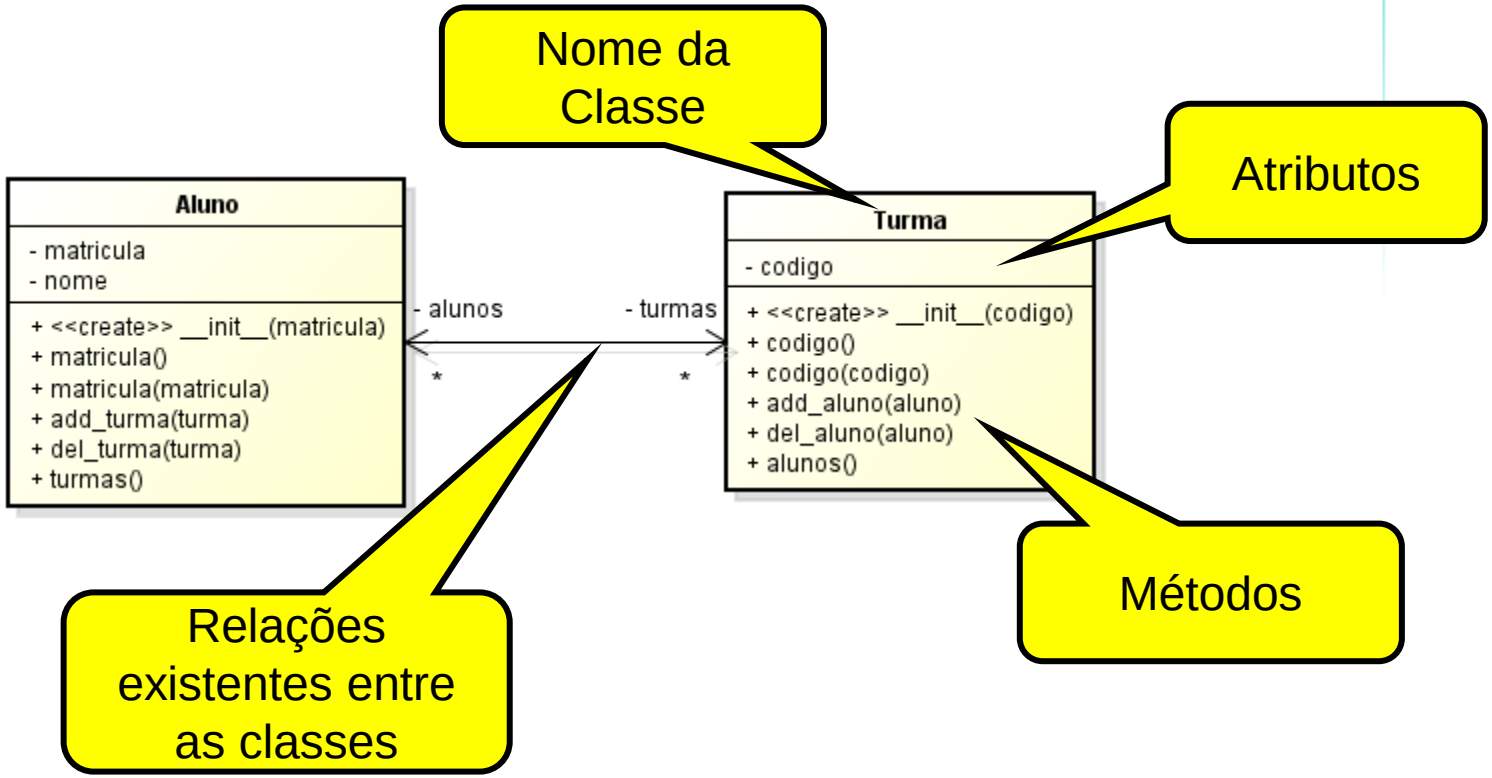
- Quando o nome do atributo for composto por mais de uma palavra, elas são concatenadas com *underline*, iniciando com letra minúscula
- Ex: data_nascimento, nome_completo, endereço_residencial ...
- Atributos encapsulados devem iniciar com **duplo *underline*** "__"

https://www.python.org/dev/peps/pep-0008/

Modelando as Classes e seus atributos



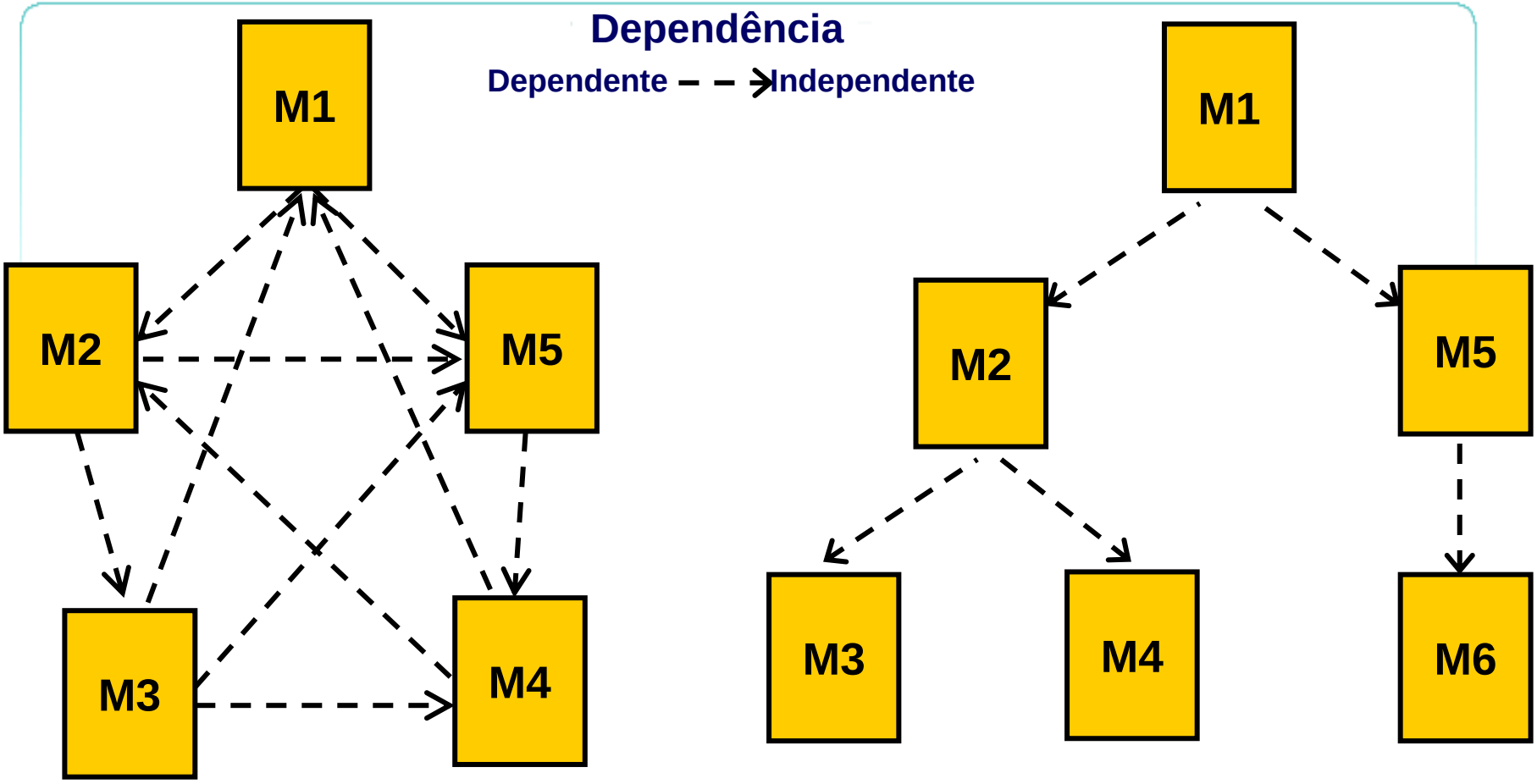
Modelando as Classes e seus atributos



Coesão e Acoplamento

- Medidas de qualidade típicas aplicadas a qualquer tipo de módulo: **sub-sistemas, classes, rotinas**, etc
- **Coesão**: quando um módulo possui um único objetivo, onde todas as partes do módulo estão alinhadas com este objetivo (medida interna)
- **Acoplamento**: grau de dependência entre os módulos (medida externa)
- É **desejável** que os módulos tenham **alta coesão** com **baixo acoplamento**

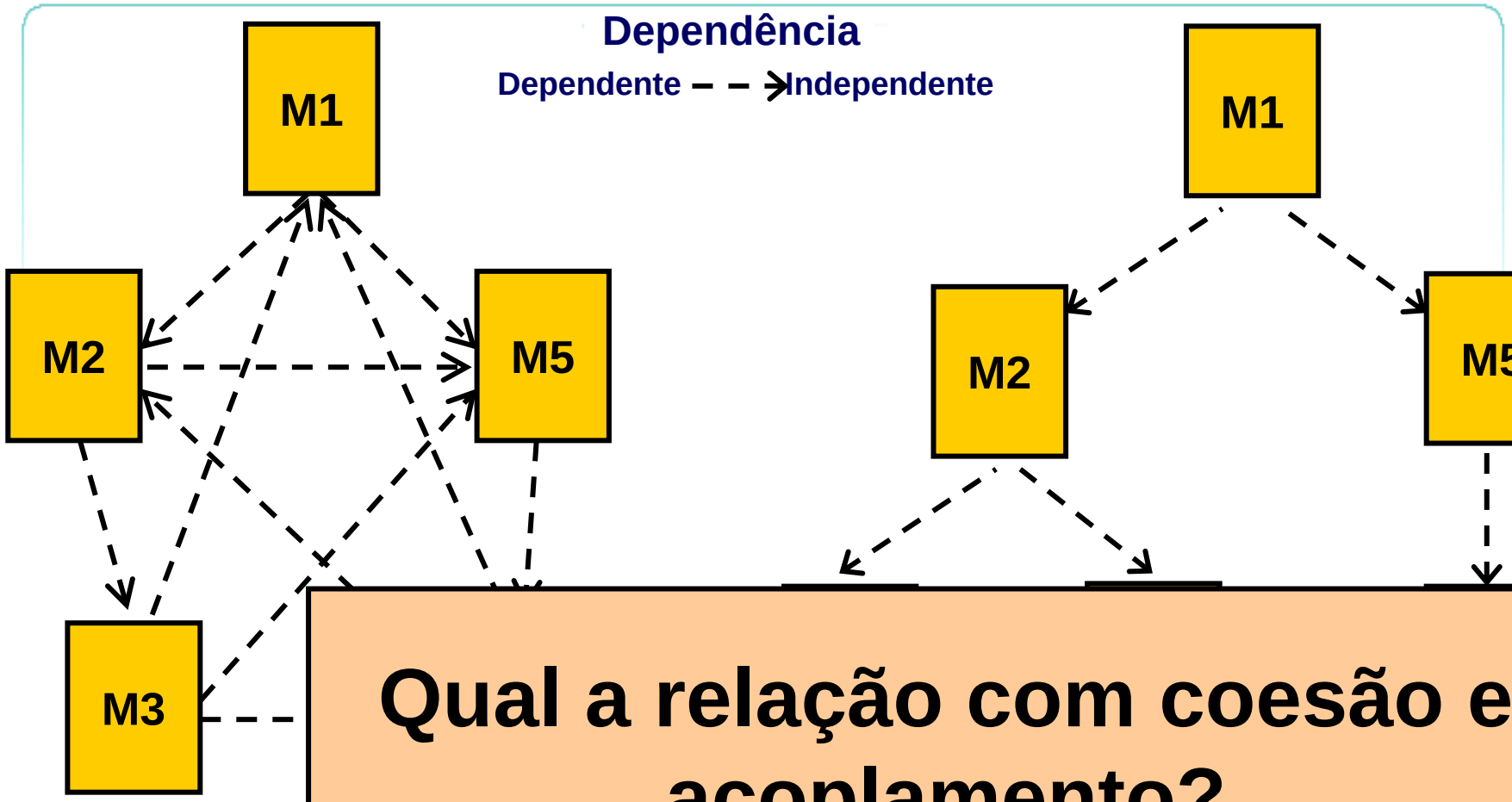
Organização dos módulos



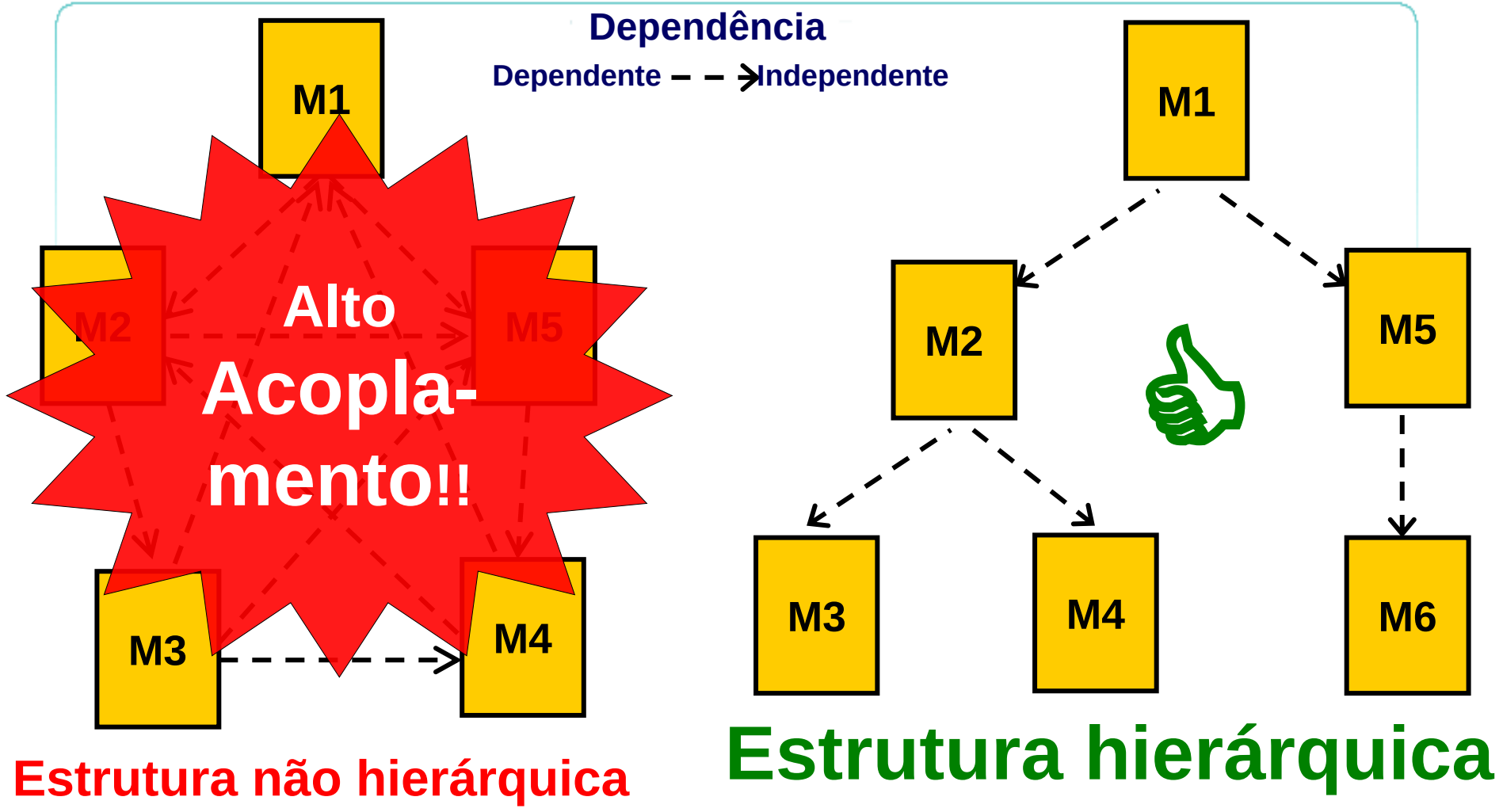
Estrutura não hierárquica

Estrutura hierárquica

Organização dos módulos



Organização dos módulos



Exercitando

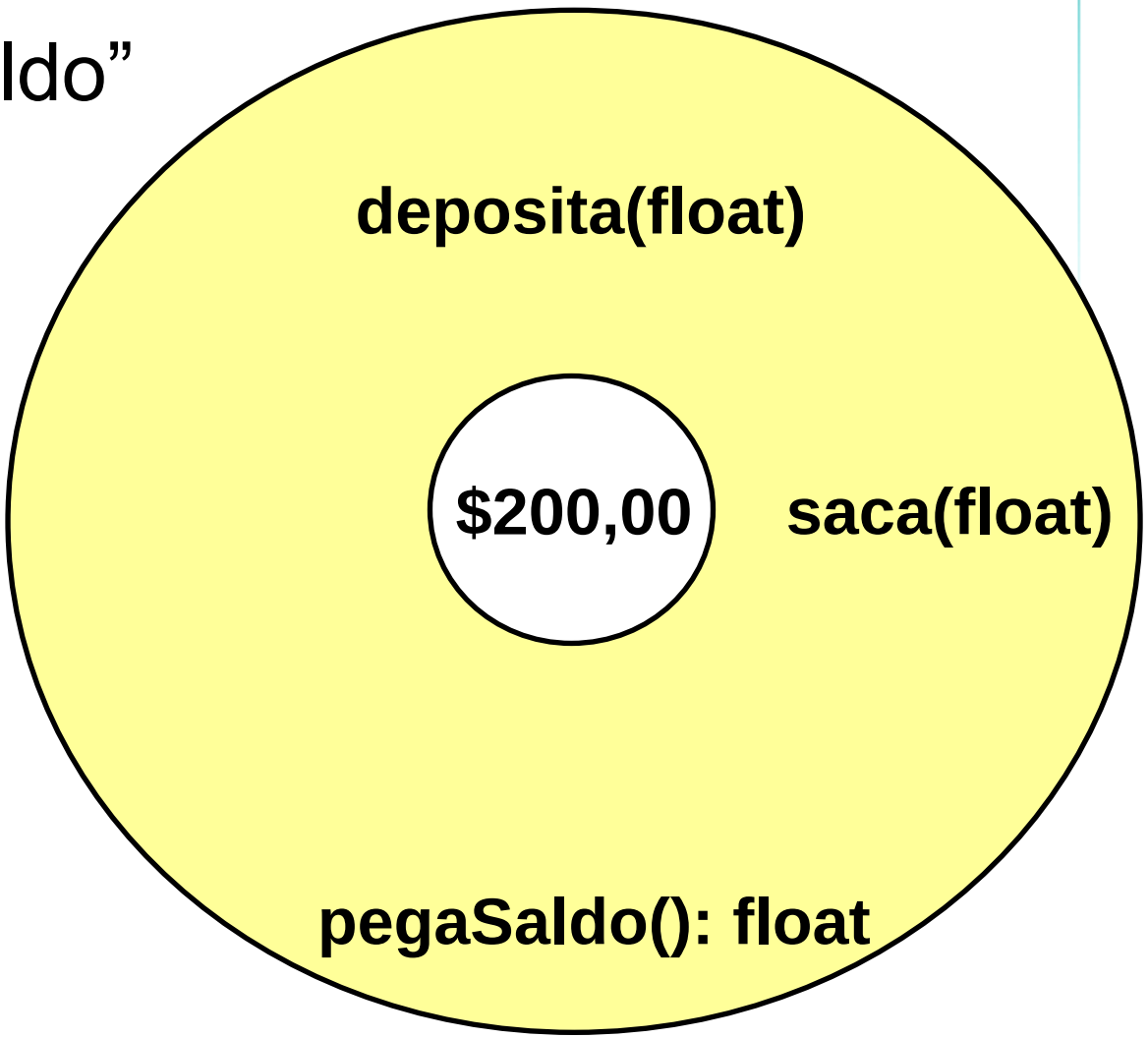
- Quais Classes você identifica nestas imagens?
- Quais os atributos dessas Classes?

Proteção aos dados

- O encapsulamento permite **esconder** o estado do objeto e a implementação de suas operações (**escondendo a informação**)
- A complexidade é **reduzida** ao escopo das chamadas de operação
- **Nem tudo precisa ser visível** → apenas aquilo que pode ser usado diretamente
- Só se deve **conhecer ou modificar o valor de um atributo** de um objeto **através de suas operações**
- Reduz a propagação de erros
- Centraliza o tratamento das regras de negócio → reduz duplicação → evita inconsistência **(DRY)**

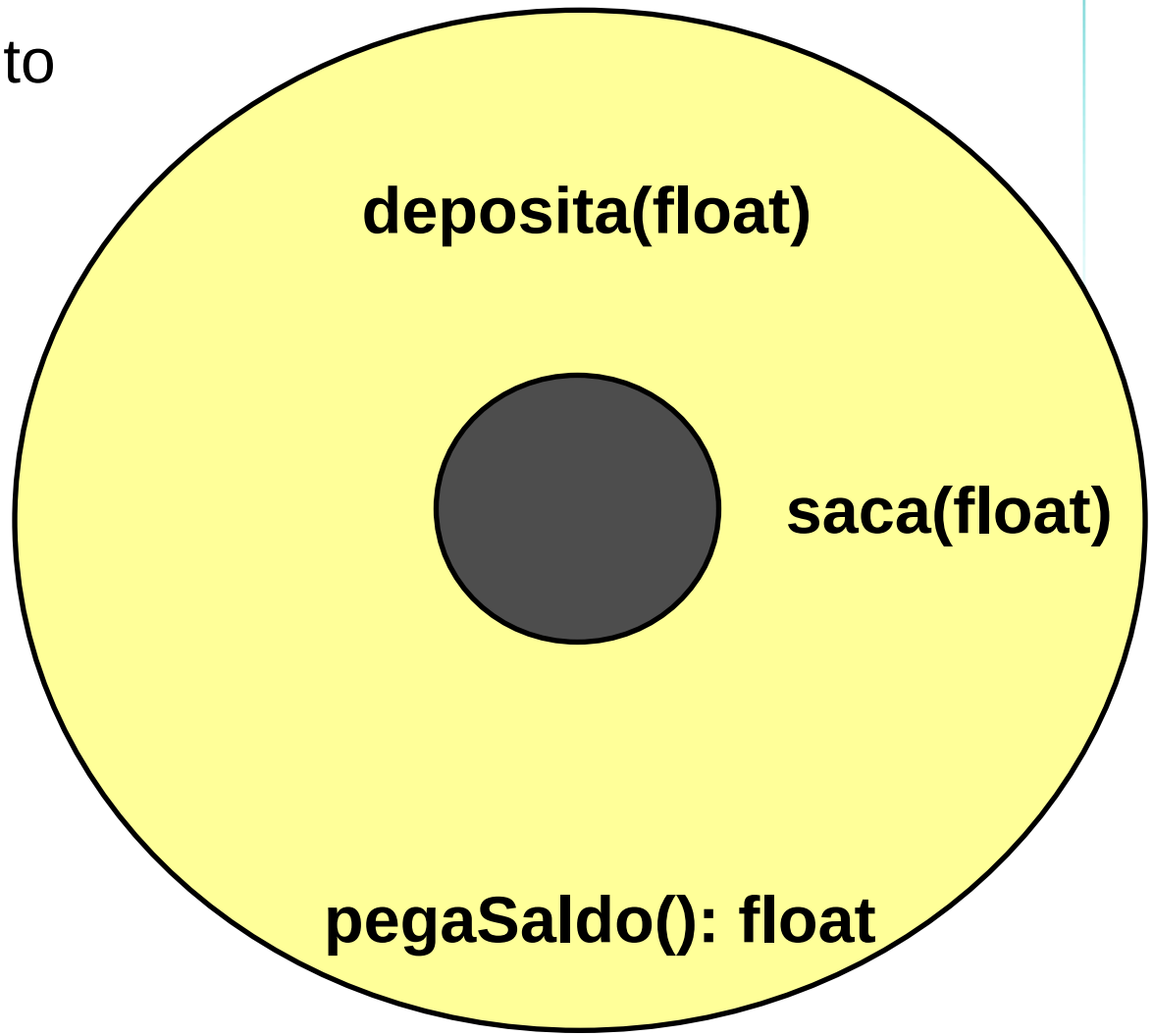
Proteção aos dados

- O atributo “saldo” precisa ser acessado diretamente?



Proteção aos dados

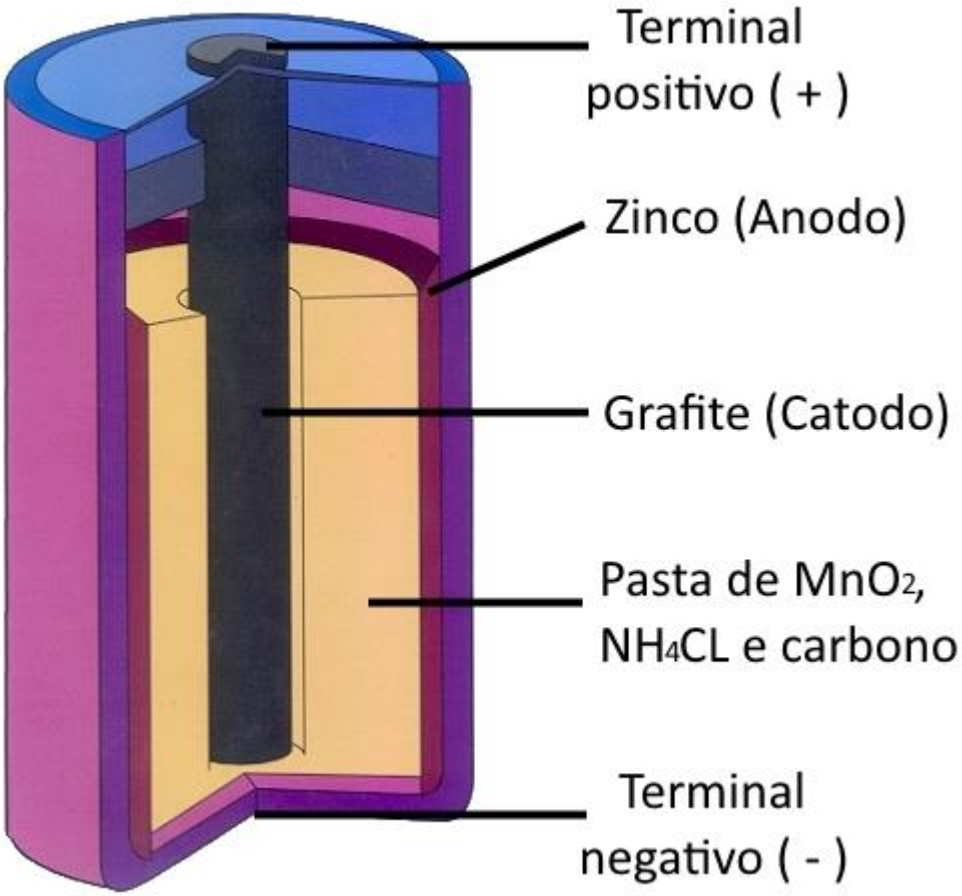
- Tudo pode ser feito através das operações disponíveis
- O atributo “saldo” fica protegido de acessos indevidos



Encapsulamento

Encapsulamento

SEM ENCAPSULAMENTO



[https://www.filipeflop.com/blog/pilhas-e-baterias-principais-tipos/]

Encapsulamento

COM ENCAPSULAMENTO

[https://www.multicoisas.com.br/]

Encapsulamento

COM ENCAPSULAMENTO

- Protege a estrutura interna de manipulações indevidas
- Protege quem está manipulando de efeitos indesejados da estrutura interna
- Define uma “interface” para o acesso externo

[https://www.multicoisas.com.br/]

Encapsulamento

COM ENCAPSULAMENTO

- Protege a estrutura interna de manipulações indevidas
- Protege quem está manipulando de efeitos indesejados da estrutura interna
- Define uma “interface” para o acesso externo

[https://www.multicoisas.com.br/]

Encapsulamento

- Um objeto é uma entidade única e indivisível
- Quando necessário, o objeto **encapsula** os dados e as operações que manipulam estes dados
- Dados só devem ser modificados pelas operações que são parte do objeto → acesso é através da **interface do objeto**
- A **interface** de um objeto é o conjunto das suas operações públicas

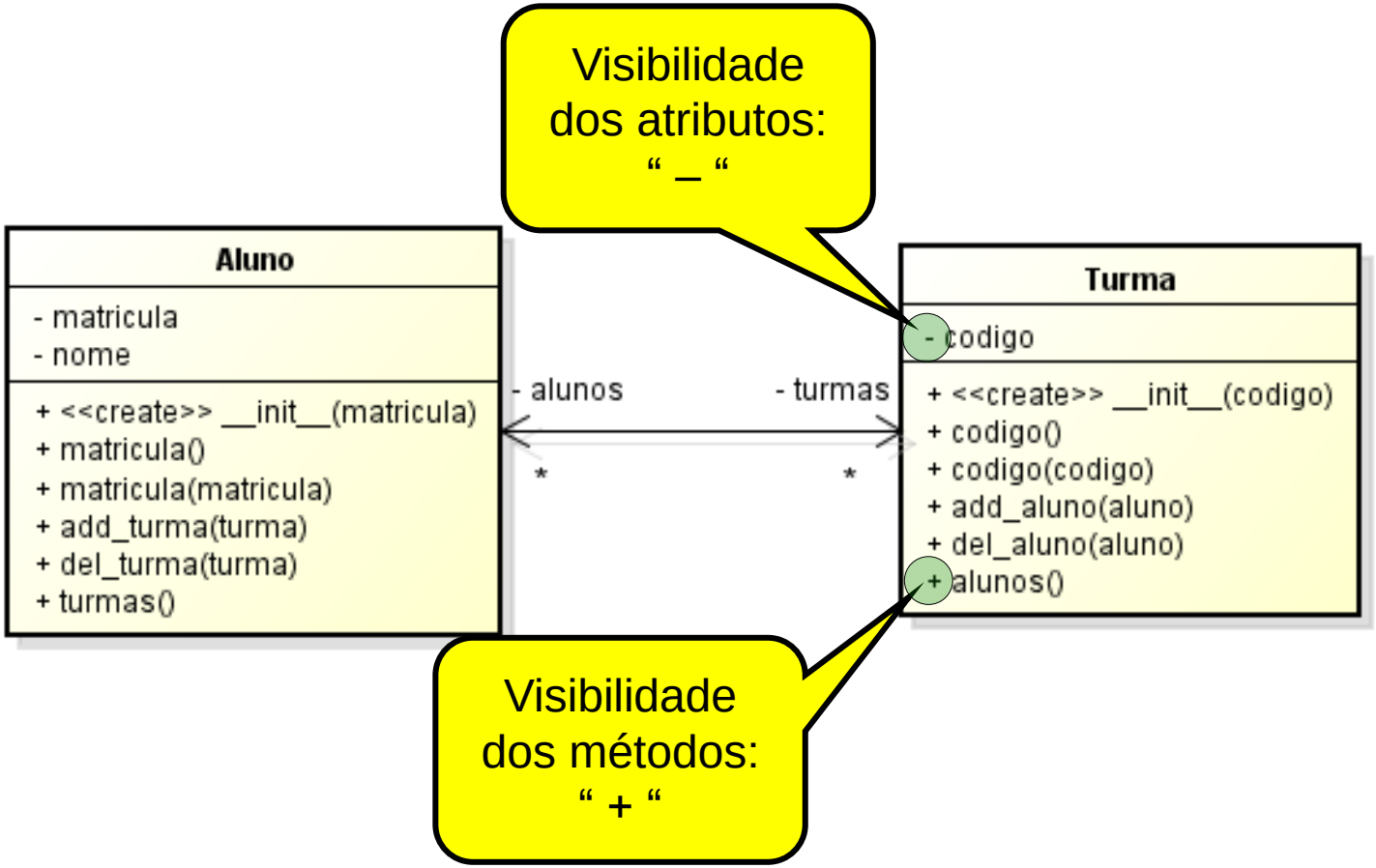
Visibilidade

Especifica como os atributos e/ou operações poderão ser acessados por outros objetos. Em Python atributos e operações **não são realmente privados**, mas podem ser **ocultos** pelo uso de algumas convenções:

- **Pública +** : (padrão em Python) atributos e operações são visíveis **dentro** da **própria classe** e para **todas as outras classes** que a importarem
- **Protegida #** : somente acessível pelas subclasses (não contemplada em Python). Convenção de _ no início do nome
- **Privada –** : atributos e operações **ficam ocultos** e só devem ser acessados na implementação da **própria classe.** Em Python usa-se: __ no início do nome para simular (Python troca **<__nome>** por **_NomeClasse__nome**)

https://docs.python.org/3.3/tutorial/classes.html#tut-private

Modelando a Visibilidade



Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = nome

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```

Uma classe em Python

```
class Pessoa:

    def __init__(s                ""):
        self.__nome      nome

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```

Nome da classe

Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = n

    @property
    def nome(self):
        return self.__

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```

self é sempre o primeiro parâmetro e se refere ao próprio objeto instanciado.

Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = nome

    @property
    def nome(sel
        return s

    @nome.setter
    def nome(sel
        self.__n
```

Construtor da classe: operação especial que permite criar novos objetos

Você pode usar o construtor para inicializações

Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = nome

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```

Atributos
Encapsulados iniciam com __

Visibilidade em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self._nome = nome

    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, nome):
        self._nome = nome
```

Indica visibilidade privada: atributos devem ser privados quando for importante garantir proteção aos dados

Uma classe em Python

```
class Pessoa:

    def __init__(self,
        self.__nome

    @property
    def nome(self):
        return self.__n

    @nome.setter
    def nome(self, nome
        self.__nome = n
```

@property
é usado quando o objetivo é retornar o valor de um atributo

@<atributo>.setter
é usado quando o objetivo é alterar o valor de um atributo

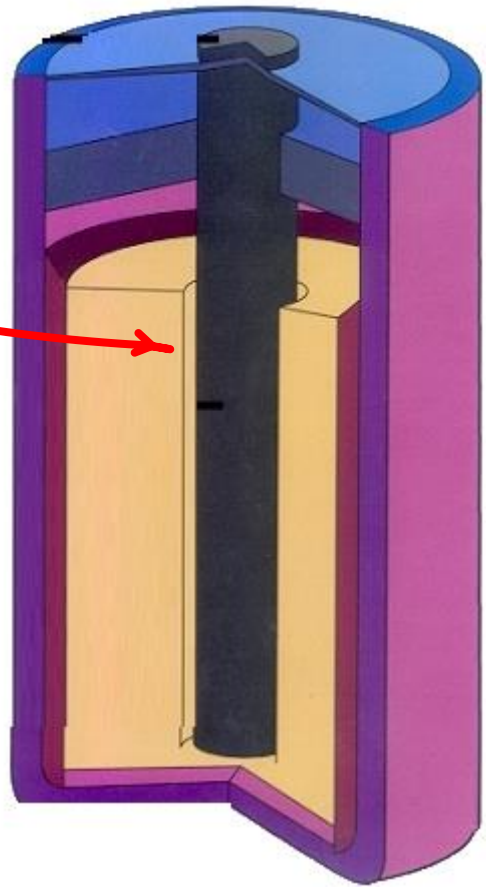
Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = nome

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```



Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = nome

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```

Uma classe em Python

```
class Pessoa:

    def __init__(self, nome=""):
        self.__nome = nome

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome
```

Objetos em Python

```
...
  uma_pessoa = Pessoa("Jean")
  uma_pessoa.nome = "Pedro"
  print(uma_pessoa.nome)
...
  outra_pessoa = Pessoa("Paulo")
  outra_pessoa.nome = "Outro Nome"
  print(outra_pessoa.nome)
```

Objetos em Python

```
...
  uma_pessoa = Pessoa("Jean")
  uma_pessoa.nome = "Pedro"
  print(uma_pessoa.nome)
...
  outra_pessoa = Pessoa("Maria")
  outra_pessoa.nome = "Ana"
  print(outra_pessoa.nome)
```

Permite declarar uma variável objeto denominada "uma_pessoa" como sendo um objeto da classe "Pessoa"

Objetos em Python

```
...
  uma_pessoa = Pessoa("Jean")
  uma_pessoa.nome = "Pedro"
  print(uma_pessoa.nom
...
  outra_pessoa = P
  outra_pessoa.nom
  print(outra_pess
```

Chamando o construtor da classe `__init__(...)` retorna uma nova instância (objeto) da classe Pessoa

Objetos em Python

```
...
  uma_pessoa = Pessoa("Jean")
  uma_pessoa.nome = "Pedro"
  print(uma_pessoa.nome)
...
  outra_pessoa = Pessoa("Paulo")
  outra_pessoa.nome = "Outro Nome“
  print(outra_pessoa.nome)
```

Objetos em Python

```
...
  uma_pessoa = Pessoa("Jean")
  uma_pessoa.nome = "Pedro"
  print(uma_pessoa.nome)
...
  outra_pessoa = Pessoa("Outro")
  outra_pessoa.nome = "Outro Nome"
  print(outra_pessoa.nome)
```

O que acontece aqui?

Objetos em Python

```
...
  uma_pessoa = Pessoa("Jean")
  uma_pessoa.nome = "Pedro"
  print(uma_pess
...
  outra_pessoa = P
  outra_pessoa.nom
  print(outra_pesso
```

O valor do nome é alterado através da operação:

```
@nome.setter
def nome(self, nome):
  ...
```

Objetos em Python

```
...
    uma_pessoa = Pessoa("Jean")
    print(uma_pessoa.__nome)
...
```

Objetos em Python

```
...
    uma_pessoa = Pessoa("Jean")
    print(uma_pessoa.__nome)
...
```

O que acontece aqui?

Objetos em Python

```
...
    uma_pessoa = Pessoa("Jean")
    print(uma_pessoa.__nome)
...
```

AttributeError: 'Pessoa' object has no attribute '__nome'

Objetos em Python

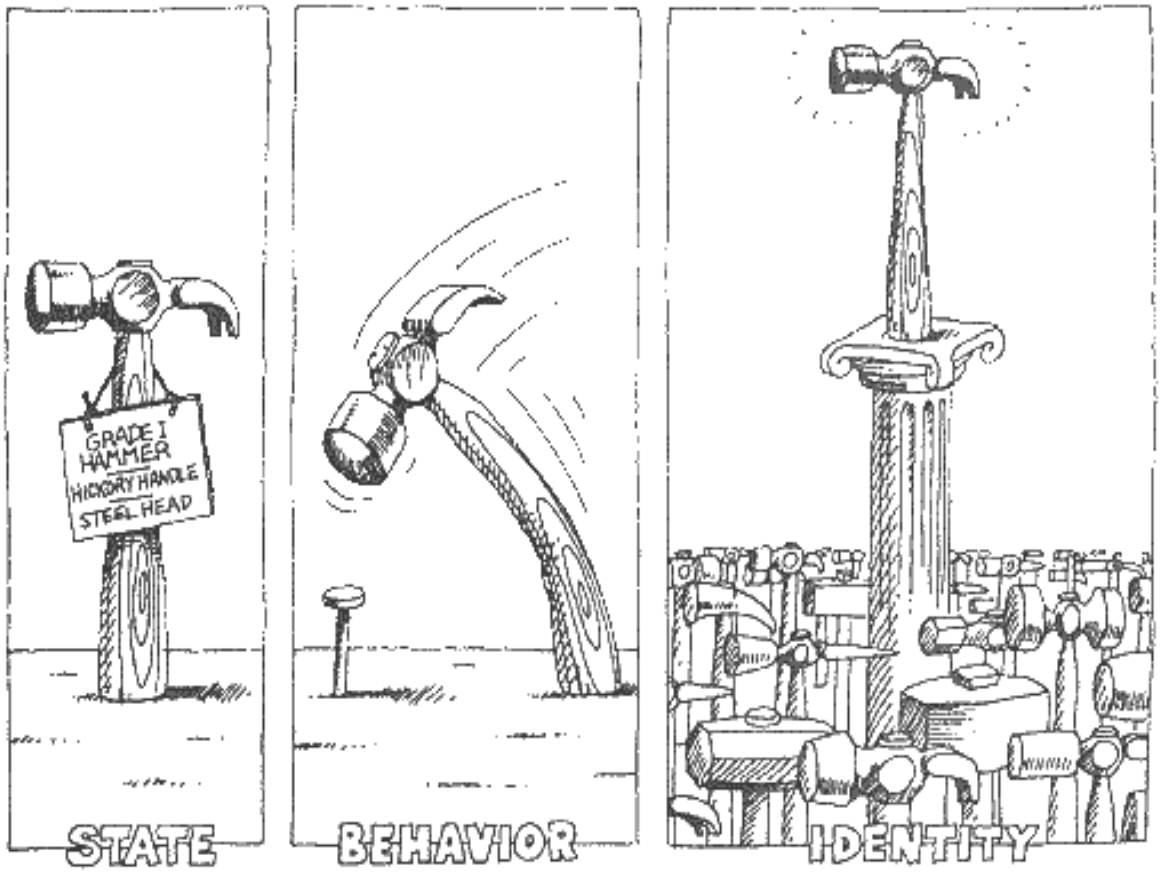
```
...
    uma_pessoa = Pessoa("Jean")
    print(uma_pessoa._Pessoa__nome)
...
```

Mas esta referência é válida!!

Então: não **há realmente atributos ou métodos privados** em Python, mas sim uma convenção de nomenclatura

Componentes de um objeto

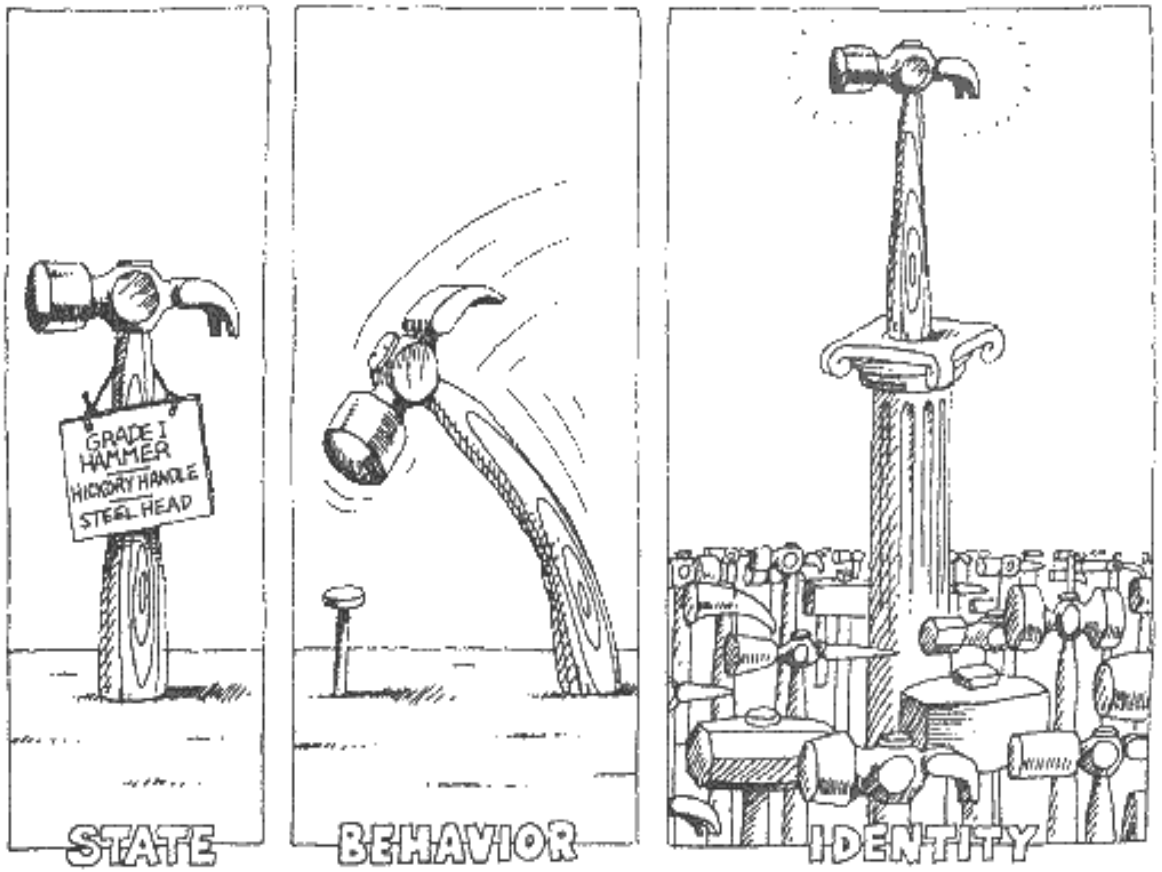
- Todo objeto tem:
  - **Estado**
  - **Comportamento**
  - **Identidade**



[BOOCH, 1994]

Componentes de um objeto

- Todo objeto tem:
  - **Estado**
  - Comportamento
  - Identidade



[BOOCH, 1994]

Estado de um objeto

- Noção do valor de um objeto em um determinado momento
- O estado de um objeto representa uma das possíveis condições em que um objeto pode existir
- O estado é representado pelos **valores** das propriedades (seus **atributos**) de um objeto **em um determinado momento**
- O estado do objeto usualmente muda ao longo do tempo

Atributos X variáveis locais

- Um **atributo** é uma **característica relevante** presente em um objeto **durante toda a vida** deste objeto.
  - O Objeto não faz sentido sem aquele atributo naquele contexto
  - Tipicamente vem do domínio do problema

- Variáveis temporárias **não devem ser declaradas como atributos**
  - Ex: variáveis que controlam laços; variáveis que guardam valores intermediários em cálculos.

Notação gráfica de estado

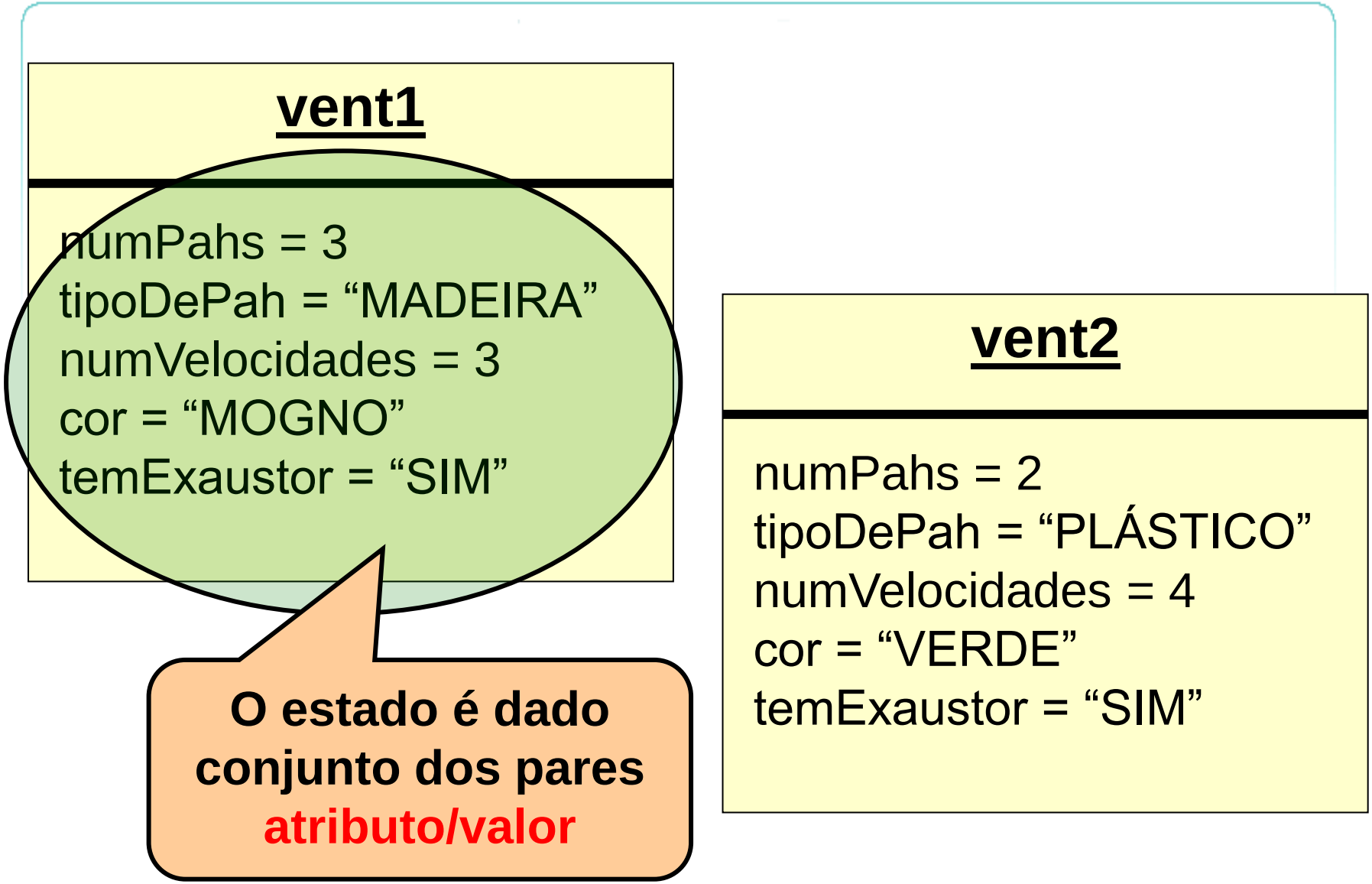
vent1

numPahs = 3
tipoDePah = “MADEIRA”
numVelocidades = 3
cor = “MOGNO”
temExaustor = “SIM”

vent2

numPahs = 2
tipoDePah = “PLÁSTICO”
numVelocidades = 4
cor = “VERDE”
temExaustor = “SIM”

Estado do objeto



Estado do objeto

vent1

numPahs = 3
tipoDePah = “MADEIRA”
numVelocidades = 3
cor = “MOGNO”
temExaustor = “SIM”

Estado parcial do objeto vent1

vent2

numPahs = 2
tipoDePah = “PLÁSTICO”
numVelocidades = 4
cor = “VERDE”
temExaustor = “SIM”

Estado do objeto

- Objeto "**Meu Ventilador**"
  - Número de pás: **3**
  - Tipo de pá: **MADEIRA**
  - Número de velocidades: **3**
  - Cor: **MOGNO**
  - Tem exaustor: **SIM**
  - Tem lustre: **SIM**

Atributos

Estado do objeto

- Objeto "**Meu Ventilador**"
  - Número de pás: 3
  - Tipo de pá: MADEIRA
  - Número de velocidades: 3
  - Cor: MOGNO
  - Tem exaustor: SIM
  - Tem lustre: SIM

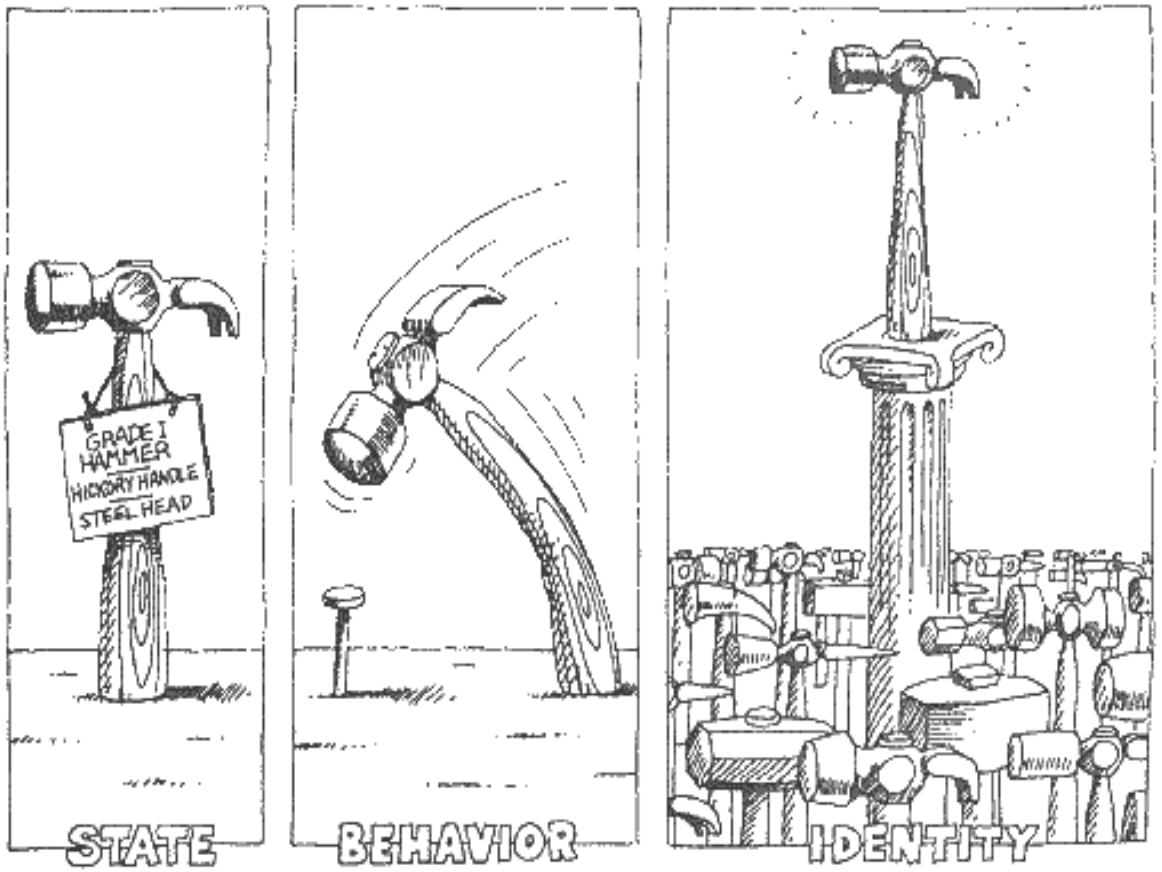
Estado atual

Estado do objeto

- Objeto "**Outro Ventilador**"
  - Número de pás: **2**
  - Tipo de pá: **PLÁSTICO**
  - Número de velocidades: **4**
  - Cor: **VERDE**
  - Tem exaustor: **NÃO**
  - Tem lustre: **SIM**

Componentes de um objeto

- Todo objeto tem:
  - Estado
  - **Comportamento**
  - Identidade



[BOOCH, 1994]

Comportamento de um objeto

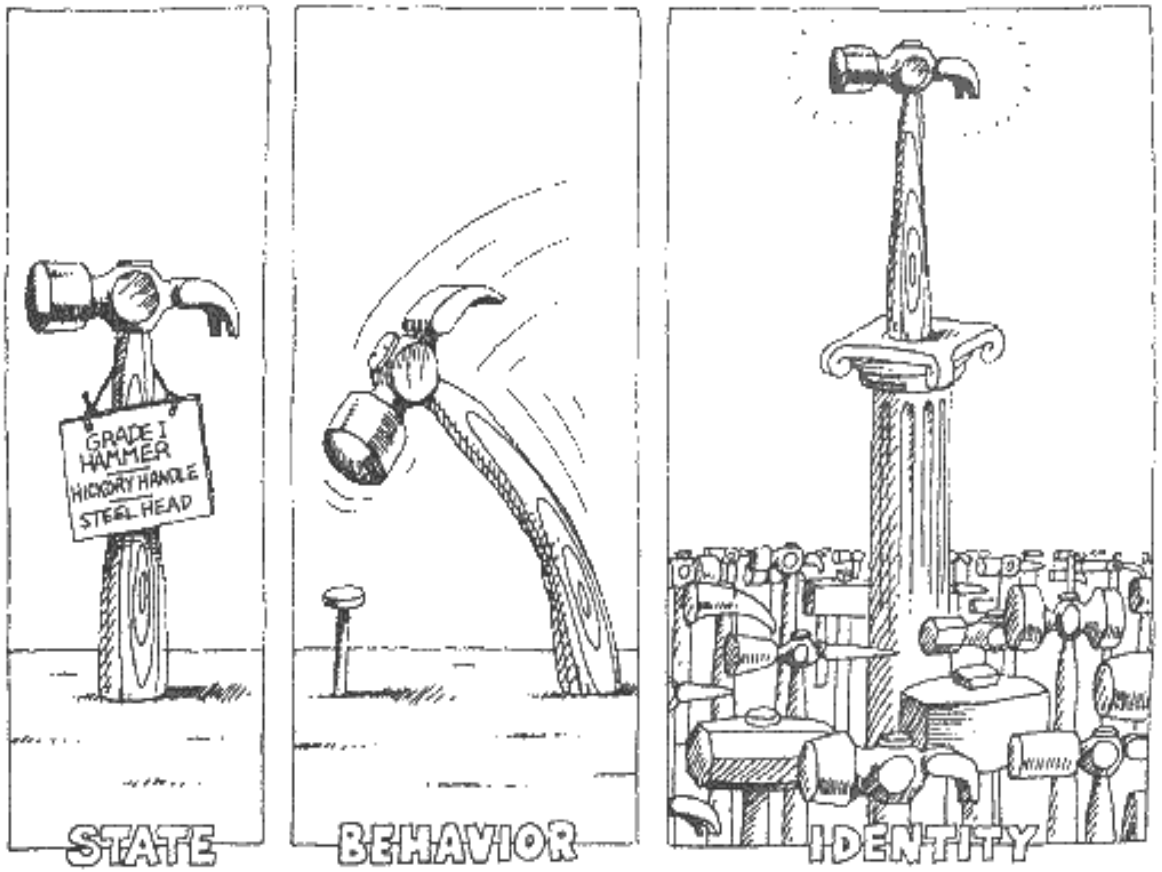
- O comportamento determina **como um objeto age e reage**: suas modificações de estado e interações com outros objetos
- O comportamento define como um objeto reage a solicitações de outros objetos
- O comportamento é determinado pelo conjunto de **operações** que o objeto pode realizar
- A **interface** de um objeto é formada pelas operações públicas de um objeto

Exercitando

- Apresente o estado de 3 objetos destas imagens
- Quais as ações que você identifica para os objetos destas imagens?

Componentes de um objeto

- Todo objeto tem:
  - Estado
  - Comportamento
  - **Identidade**



[BOOCH, 1994]

Identidade de um Objeto

SN#123345

SN#453425

SN#434247

SN#223450

SN#112112

SN#332232

SN#098934

SN#654874

SN#457778

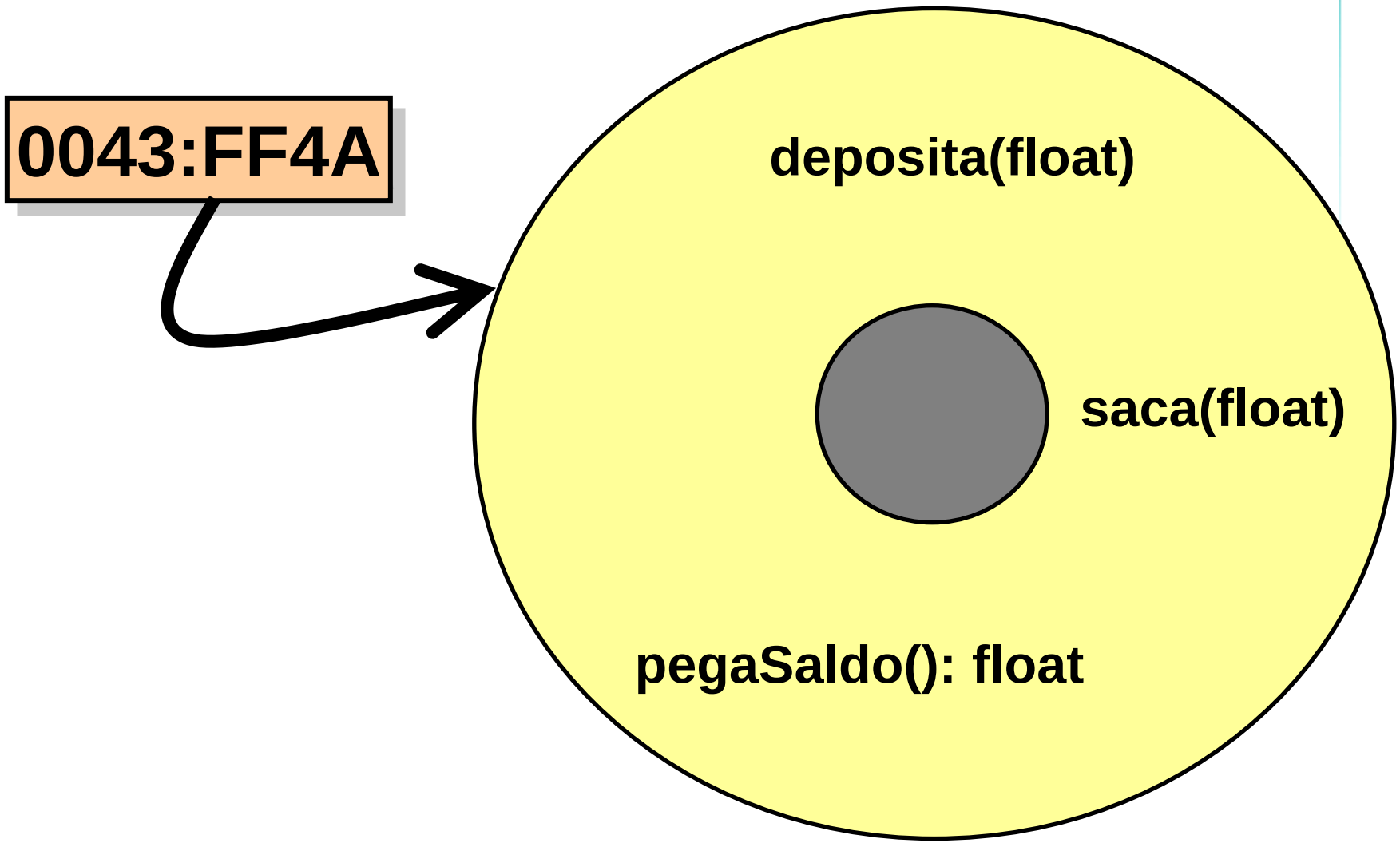
SN#988524

Identidade de um objeto

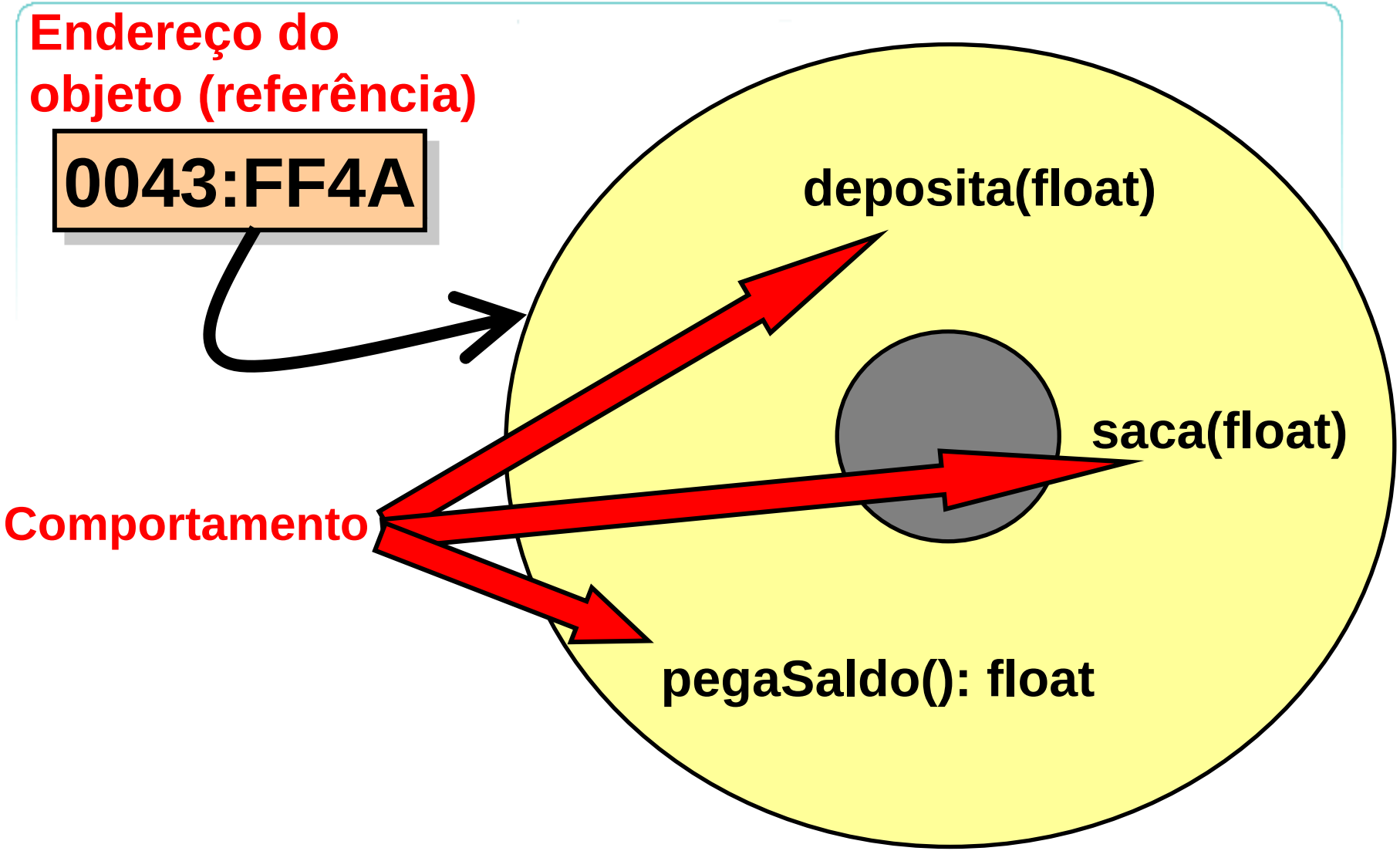
- Todo objeto tem sua própria identidade
- **Identifica unicamente*** um objeto (independentemente do seu estado)
- Identificador do objeto (***object handle***)

*** Não significa que o objeto precisa ter algum atributo específico para garantir a identidade única → objetos podem ter estados idênticos e ainda sim, serem objetos únicos**

Identidade do objeto: exemplo



Identidade do objeto: exemplo



Programação *Chuck Norris* em Python

Um código Python comum executa mais rápido quando Chuck Norris assiste a execução.

Chuck Norris corrige um valor incorreto simplesmente encarando a variável.

Quanto Chuck Norris olha com concentração um trecho de código ... ele consegue ver a própria nuca.

O código de Chuck Norris é tão rápido que, durante os testes em um laboratório, ele quebrou a velocidade da luz matando 37 pessoas.

Chuck Norris inventou uma nova versão de Python que roda em máquinas de escrever e com alto desempenho (!)

Referências

THIRY, M. Apresentações de aula. Univali, 2014.

ALCHIN, Marty. Pro Python. New York: Apress, 2010. Disponível em: <https://link.springer.com/book/10.1007%2F978-1-4302-2758-8#about>

HALL, Tim; STACEY, J. P. Python 3 for absolute beginners. Apress, 2010. Disponível em: <https://link.springer.com/book/10.1007%2F978-1-4302-1633-9>

BOOCH, G., Object-Oriented Design. Benjamin/Cumminggs Pub. 1998.

WAZLAWICK, Raul S. Introdução a Algoritmos e Programação com Python. São Paulo: Elsevier, 2017.

WAZLAWICK, Raul S. Análise e Projeto de Sistemas de Informação Orientados a Objetos. São Paulo: Campus. 2004

Agradecimento

Agradecimento especial ao prof. Marcello Thiry pelo material cedido.

marcello.thiry@gmail.com

Licença

creative commons
COMMONS DEED

Atribuição-Uso-Não-Comercial-Compartilhamento pela Licença 2.5 Brasil

Você pode:
- copiar, distribuir, exibir e executar a obra
- criar obras derivadas

Sob as seguintes condições:
Atribuição — Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.
Uso Não-Comercial — Você não pode utilizar esta obra com finalidades comerciais.
Compartilhamento pela mesma Licença — Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

Para ver uma cópia desta licença, visite http://creativecommons.org/licenses/by-nc-sa/2.5/br/ ou mande uma carta para Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.