Sistemas Distribuídos e Mobile

Revisão de POO

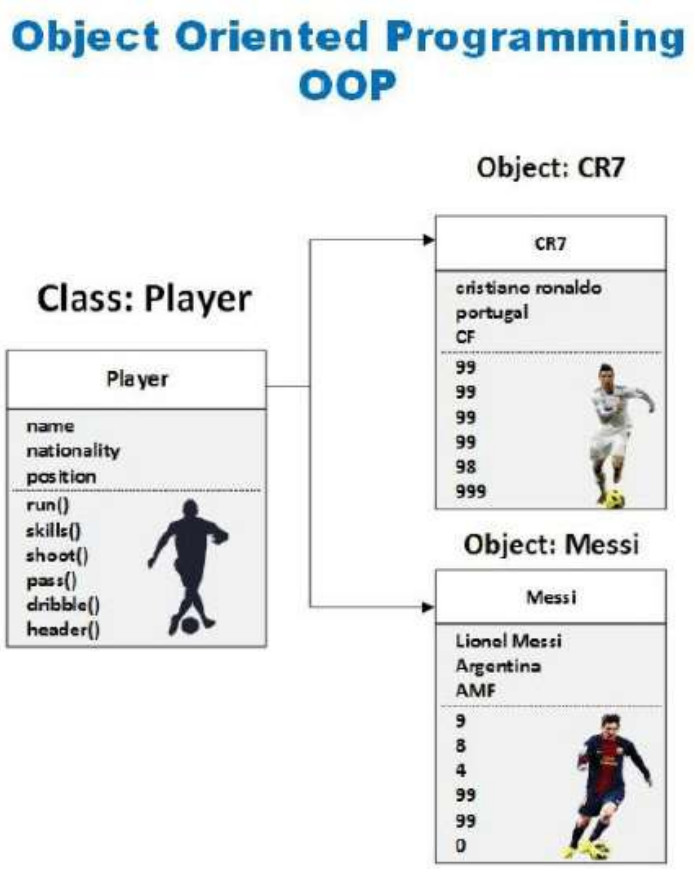
Prof. Dr. Fernando Kakugawa

frkakugawa@anhembi.br

Universidade Anhembi Morumbi

Classe

- Uma Classe representa um grupo de elementos com:
  - Características (atributos)
  - Habilidades (métodos)
- em comum!



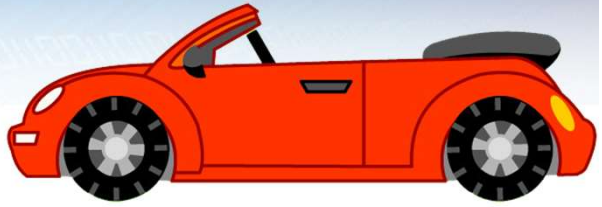
How to learn OOP using football

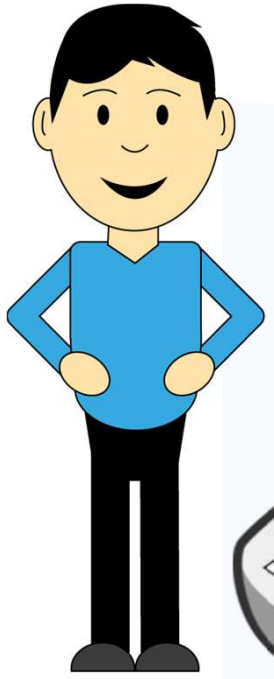
Classe

- Como agrupar as figuras?



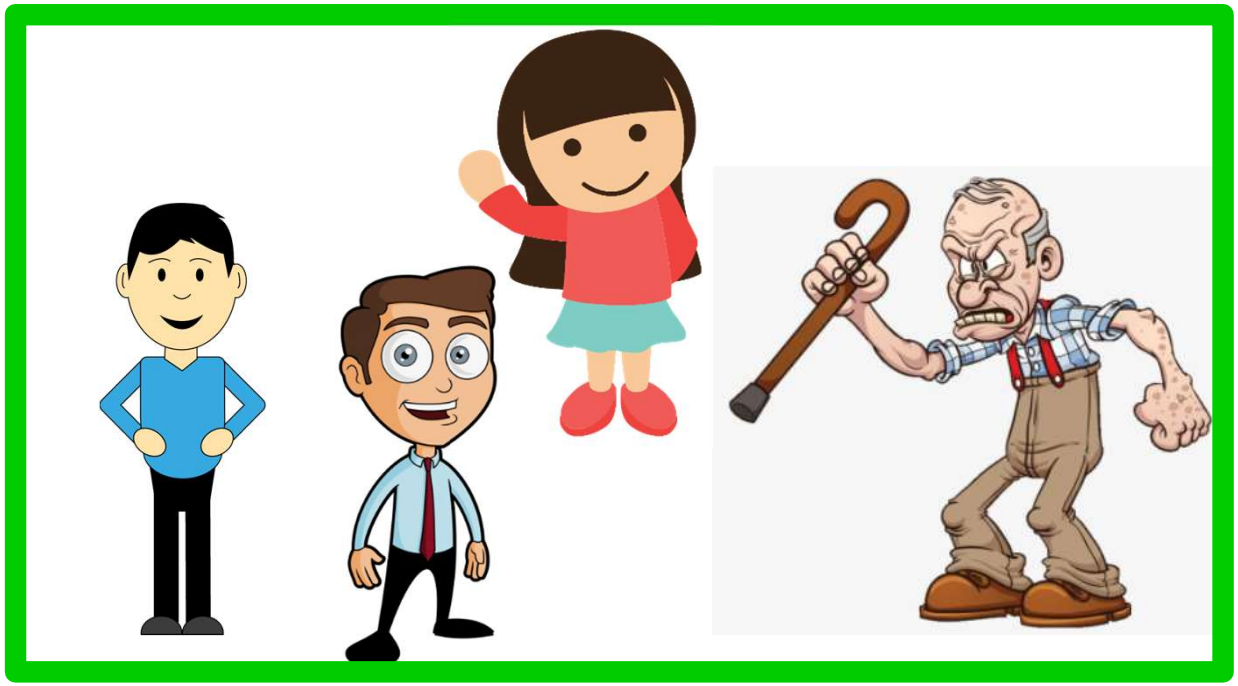








Classe

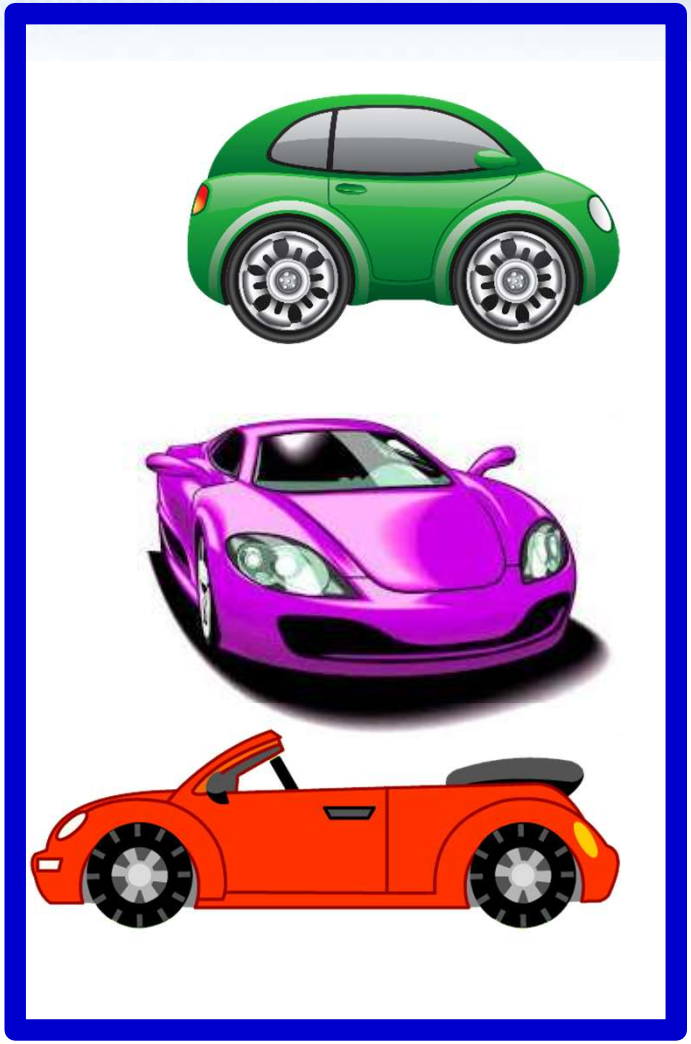


Classe

- Características
  - Tipo de tela
  - Teclado
  - Conectividade
- Habilidades
  - Ligar
  - Enviar mensagens
  - Alarme

Classe

- Características
  - Cor
  - Marcha
  - Motor
  - Velocidade
- Habilidades
  - Frear
  - Acelerar
  - Trocar marcha



Classe

- Características
  - Nome
  - Idade
  - Altura
  - Peso
- Habilidades
  - Dormir
  - Ler
  - Pensar

Definição de Classes

- Em Java, classes são definidas através do uso da palavra-chave **`class`**.
- Sintaxe para definir uma classe:

```
[modificador] class NomeDaClasse {
// corpo da classe...
}
```

  - Após a palavra-chave **`class`**, segue-se o nome da classe, que deve ser um identificador válido para a linguagem.
  - [modificador] é opcional; se presente, pode ser uma combinação de **`public`** e **`abstract`** ou **`final`**.
- O modificador **`abstract`** indica que nenhum objeto dessa classe pode ser instanciado.
- O modificador **`final`** indica que a classe não pode ser uma superclasse (uma classe não pode herdar de uma classe final)

Construtor

- Um construtor é um método especial, definido para cada classe.
  - Determina as ações associadas à inicialização de cada objeto criado.
  - É invocado toda vez que o programa instancia um objeto dessa classe.
  - A assinatura de um construtor diferencia-se das assinaturas dos outros métodos por não ter nenhum tipo de retorno (nem mesmo void).
  - O nome do construtor deve ser o próprio nome da classe.
  - O construtor pode receber argumentos, como qualquer método.
  - Toda classe tem pelo menos um construtor sempre definido.

Construtor

```
public class Pessoa {
	private String nome;
	private int idade;

	public Pessoa(String nome, int idade) {
		this.nome = nome;
		this.idade = idade;
	}

	public Pessoa() {
		this(“Bruno”, 20);
	}
}
```

Instância

- A instanciação é um processo por meio do qual se realiza a cópia de um objeto (classe) existente.
- Uma classe, a qual tem a função de determinar um tipo de dado, deve ser instanciada para que possamos utilizá-la.
- Sendo assim, devemos criar sua instância, a qual definimos como sendo um objeto referente ao tipo de dado que foi definido pela classe.

Instância

- A criação do objeto é feita pelo operador **new**
  - `<Nome da Classe> <nome do Objeto> = new <Nome da Classe>(<argumentos>);`
- Exemplos
  - `Pessoa pedro = new Pessoa(“Pedro”, 32);`
  - `Pessoa p1 = new Pessoa();`

Objeto

- Cada objeto se diferencia um do outro pelo valor de seus atributos

- Altura: 1.2
- Idade: 4
- Peso: 26

- Altura: 1.6
- Idade: 95
- Peso: 45

Referencia this

- É uma referência a um objeto
- Quando um método de uma classe faz referência a outro membro dessa classe para um objeto específico dessa classe, como Java assegura que o objeto adequado recebe a referência?
  - Cada objeto tem uma referência a ele próprio - chamada de referência **`this`**
  - Utiliza-se a referência **`this`** implicitamente para fazer referências às variáveis de instância e aos métodos de um objeto

Referencia this

- Exemplos de uso de **`this`**
  - A palavra-chave **`this`** é utilizada principalmente em dois contextos:
    - Diferenciar atributos de objetos, de parâmetros ou variáveis locais de mesmo nome;
    - Acessar o método construtor a partir de outros construtores.
- Utilizar **`this`** explicitamente pode aumentar a clareza do programa em alguns contextos em que **`this`** é opcional.

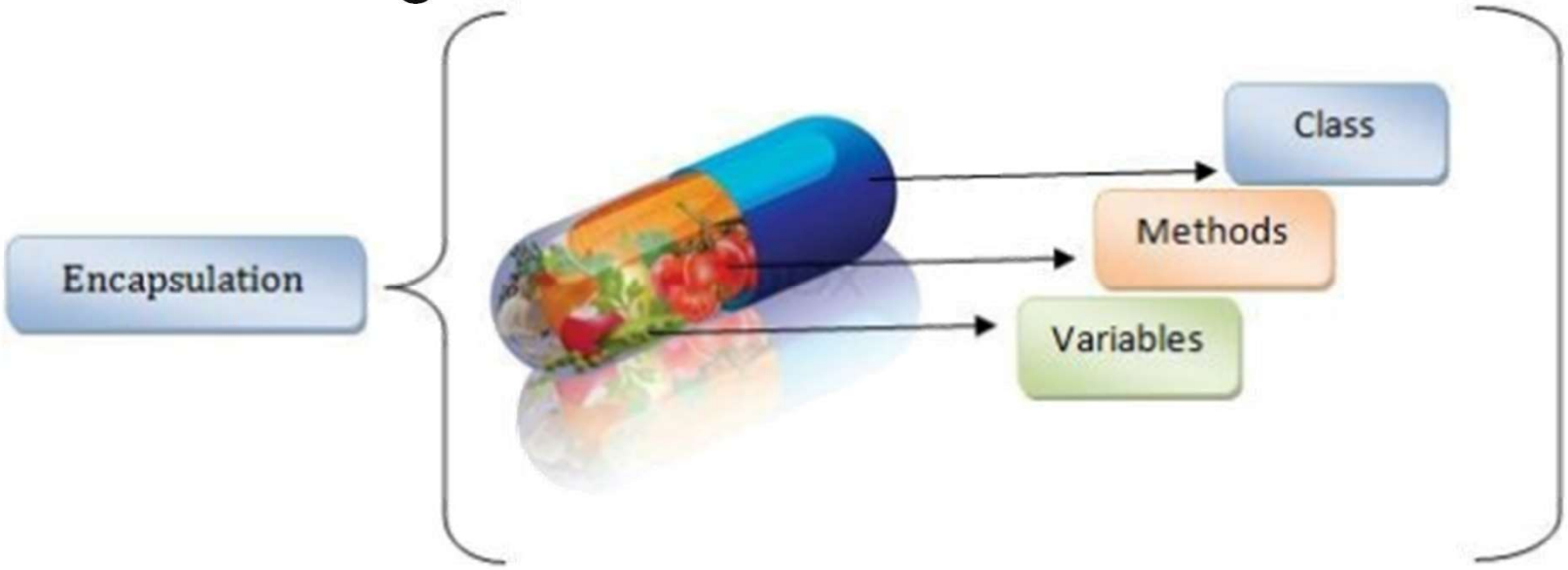
Referencia this

- Esse exemplo ilustra esses dois usos:

```
public class EsteExemplo {
	int x;
	int y;
	// exemplo do primeiro caso:
	public EsteExemplo(int x, int y){
		this.x = x;
		this.y = y;
	}
	// exemplo do segundo caso:
	public EsteExemplo (){
		this(1, 1);
	}
}
```

Encapsulamento

- Permite com que os detalhes internos de funcionamento dos métodos permaneçam ocultos para os objetos
  - Protege o acesso aos valores dos atributos
  - Métodos “get” e “set”



Encapsulamento

- Public
  - Este é o modificador menos restritivo.
  - Métodos e atributos podem ser acessados pela sua classe e por todas as outras.
- Private
  - Este é o modificador mais restritivo e o mais comum.
  - Se utilizar este modificador com um atributo ou método, ele só pode ser acessado pela classe em que pertence. Sub-classes ou outras classes não pode acessar o atributo ou método declarado como private.
- Protected
  - Métodos e atributos podem ser acessados:
    - sua classe, classes do mesmo pacote e por suas sub-classes.
- Sem modificadores
  - Pode ser acessado pela sua classe e por todas as classes que estão no mesmo pacote.

Exercício 1

- Uma classe Automóvel com os seguintes atributos:
  - Nome do proprietário
  - Modelo
  - Placa
  - Ano
- É possível alterar o nome do proprietário e imprimir os dados do automóvel.
- Fazer uma classe que possibilite a transferência de proprietários.

Associação

- É um tipo de relacionamento entre classes
- Objetos de uma classe estão conectados a objetos de outra classe (ou da mesma classe)
- Representam relacionamento “tem um”
  - Livro “tem um” capítulo
  - Carro “tem uma” roda

Vetor

- Um vetor pode conter um conjunto de valores de um mesmo tipo
- Inclusive objetos

Índice	0	1	2	3	4	5	6	7	8	9
notas	10	8,5	6,0	4,0	5,6	7,5	9,2	3,7	1,8	9,5

Vetor de objetos

- É possível criar um vetor para armazenar um conjunto de objetos de uma mesma classe
- Cada elemento do vetor representa um objeto desta classe

Vetor de objetos

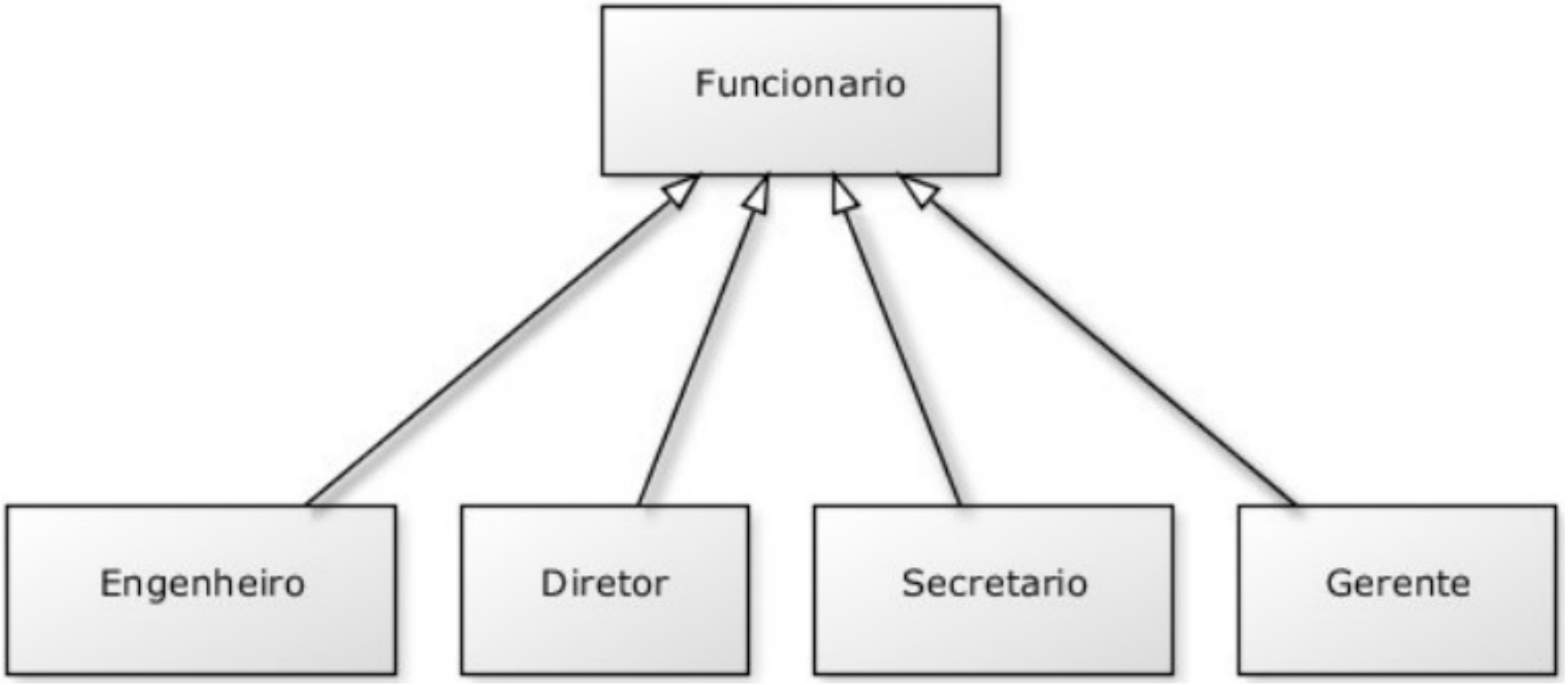
- Declaração de um vetor de objetos:
  - `Pessoa[] p = new Pessoa[6];`
- Acesso a um atributo:
  - `p[0].setNome(“Pedro”);`
  - `p[1].setNome(“Maria”);`

Índice	0	1	2	3	4	5	6	7	8	9
pessoa	obj	obj	obj	obj	obj	obj	obj	obj	obj	obj

Herança

- Permite a criação de novas classes a partir de outras previamente criadas.
  - novas classes são chamadas de subclasses, ou classes derivadas;
  - classes já existentes são chamadas de superclasses, ou classes base.
- Deste modo é possível criar uma hierarquia dessas classes, tornando, assim, classes mais amplas e classes mais específicas.
- Uma subclasse herda métodos e atributos de sua superclasse; apesar disso, pode escrevê-los novamente para uma forma mais específica de representar o comportamento do método herdado.
- Representam relacionamento “é um”
  - Gerente “é um” funcionário
  - Leão “é um” animal

Herança



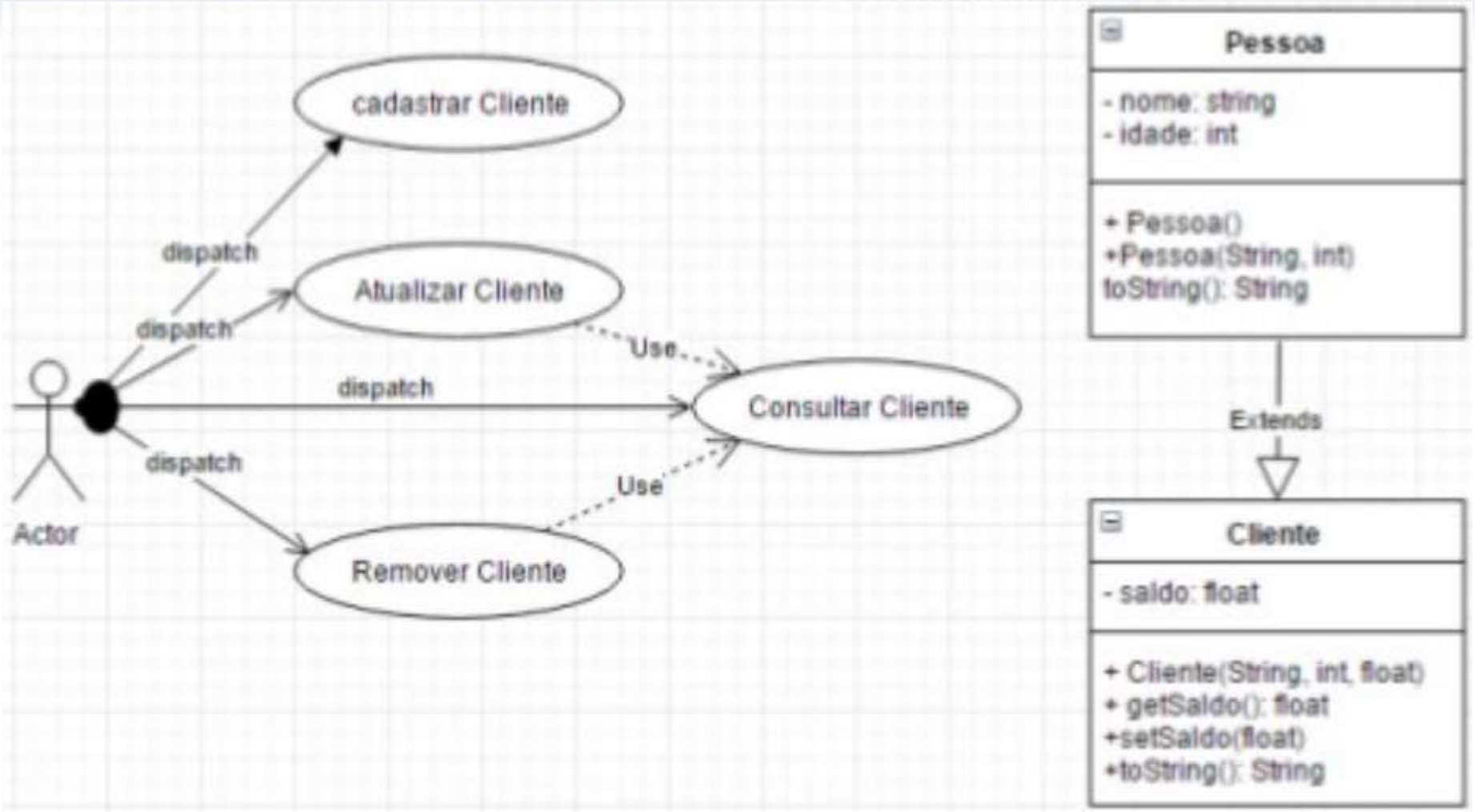
Herança

```
public class Funcionario {
… }

public class Engenheiro extends  Funcionario{
… }

public class Diretor extends Funcionario{
… }

public class Secretario extends  Funcionario{
… }

public class Gerente extends Funcionario{
… }
```

Exercício 2

- Um gerente de banco precisa de um software para cadastrar os seus clientes.
- Neste cadastro será possível:
  - inserir, remover, alterar e consultar os dados desse cliente.

Exercício 2


cadastrar Cliente
Atualizar Cliente
Consultar Cliente
Remover Cliente
Actor
dispatch
dispatch
dispatch
dispatch
Use
Use
Pessoa
- nome: string
- idade: int
+ Pessoa()
+Pessoa(String, int)
toString(): String
Extends
Cliente
- saldo: float
+ Cliente(String, int, float)
+ getSaldo(): float
+setSaldo(float)
+toString(): String

Material elaborado por:

Prof. Dr. Fernando Kakugawa

frkakugawa@anhembi.br