



2. ENCAPSULAMENTO

PROGRAMAÇÃO ORIENTADA AOS OBJETOS

Desenvolvido por:

Carlos Urbano
Catarina Reis
José Magno
Marco Ferreira
Ricardo Antunes

2.1. INVOCAÇÃO DE FUNCIONALIDADES

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

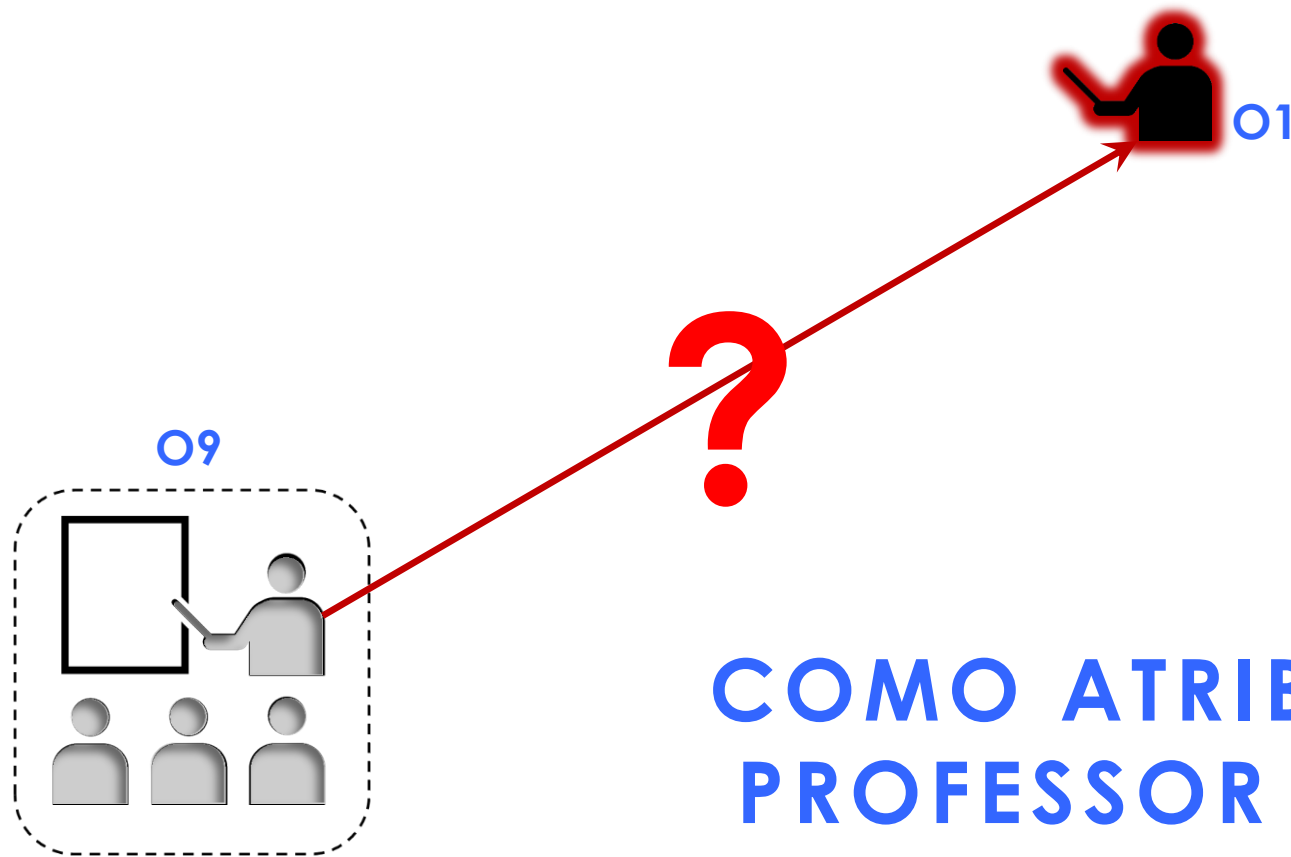
2.3. DEFINIÇÃO DE ENCAPSULAMENTO

2.4. EXEMPLO

2.5. OVERLOADING DE MÉTODOS

2.1. INVOCAÇÃO DE FUNCIONALIDADES

3



**COMO ATRIBUIR UM
PROFESSOR A UMA
AULA?**

2.1. INVOCAÇÃO DE FUNCIONALIDADES

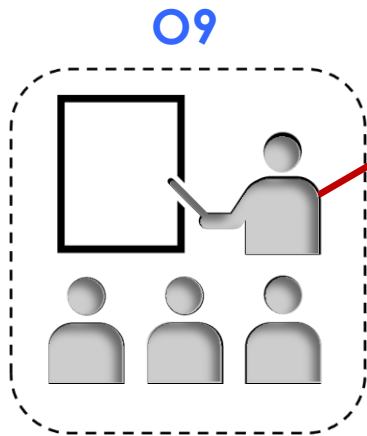
4

ENVIAR MENSAGEM

«ATRIBUIR O1»

À AULA

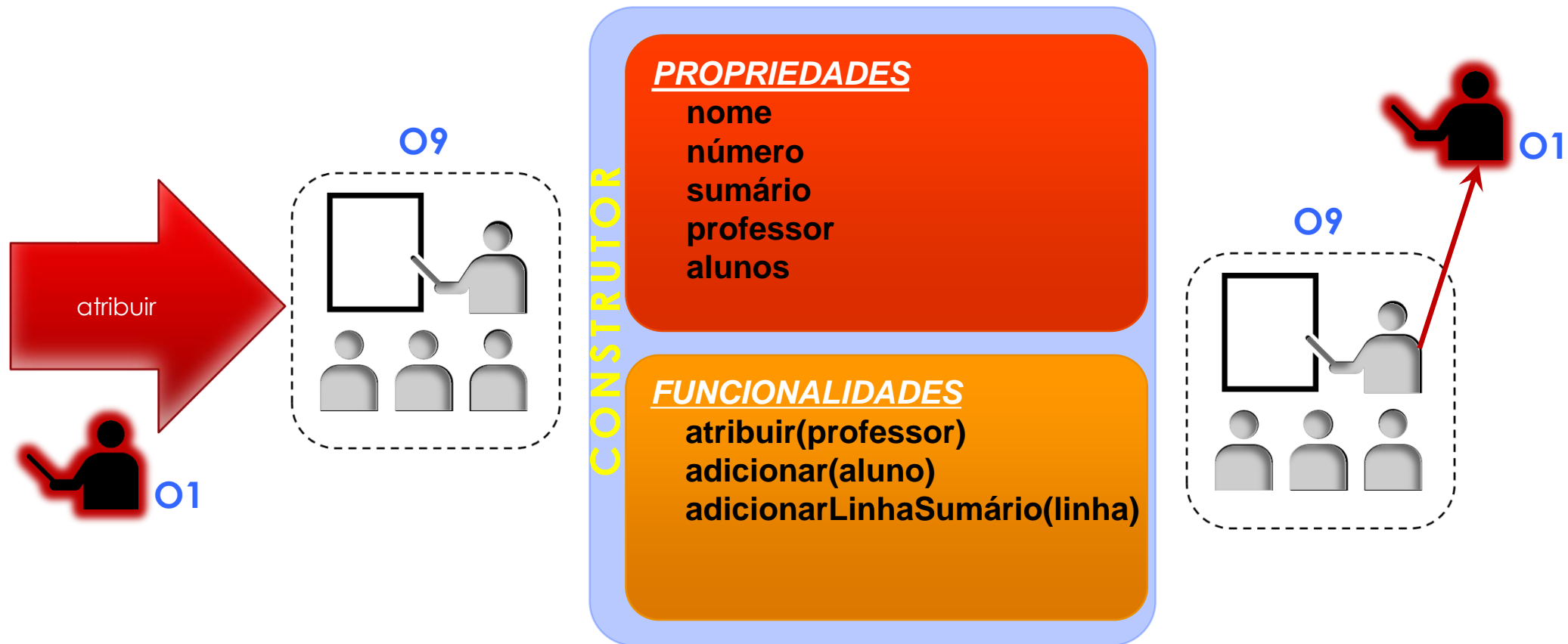
O9



2.1. INVOCAÇÃO DE FUNCIONALIDADES

5

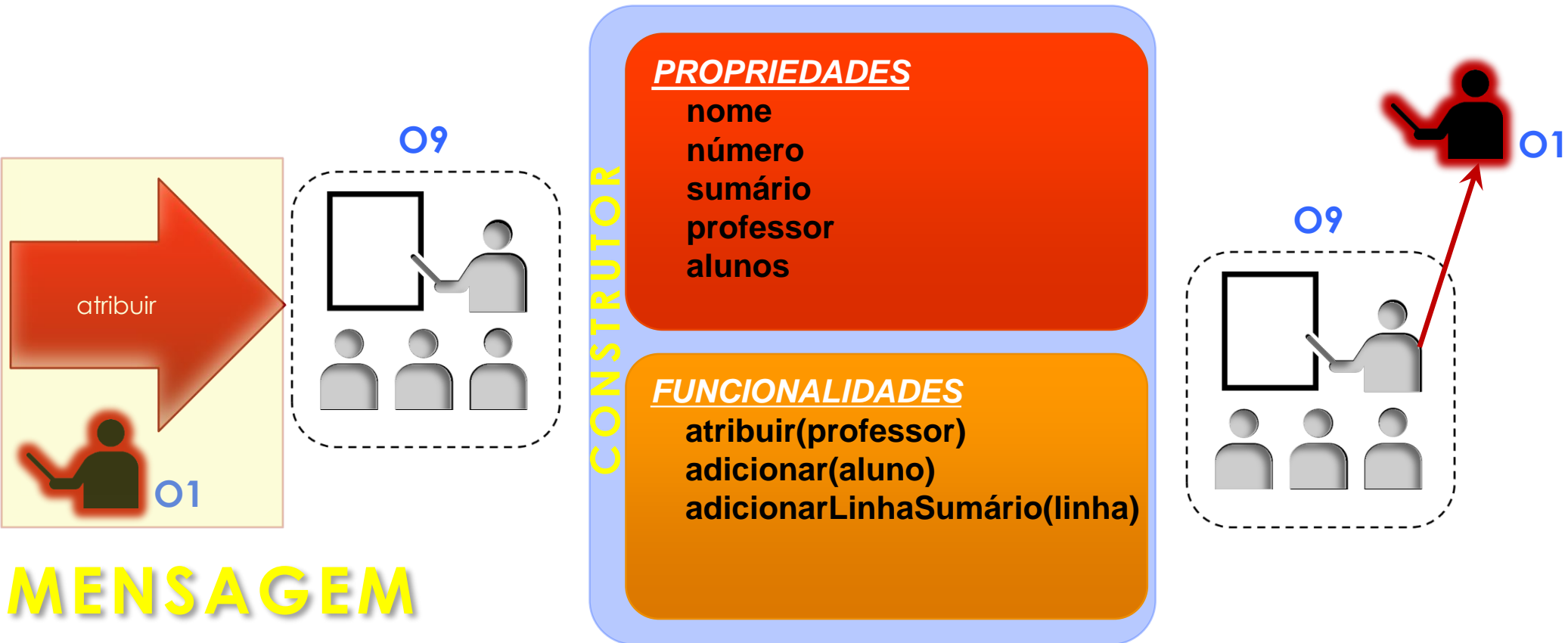
aula



2.1. INVOCAÇÃO DE FUNCIONALIDADES

6

aula

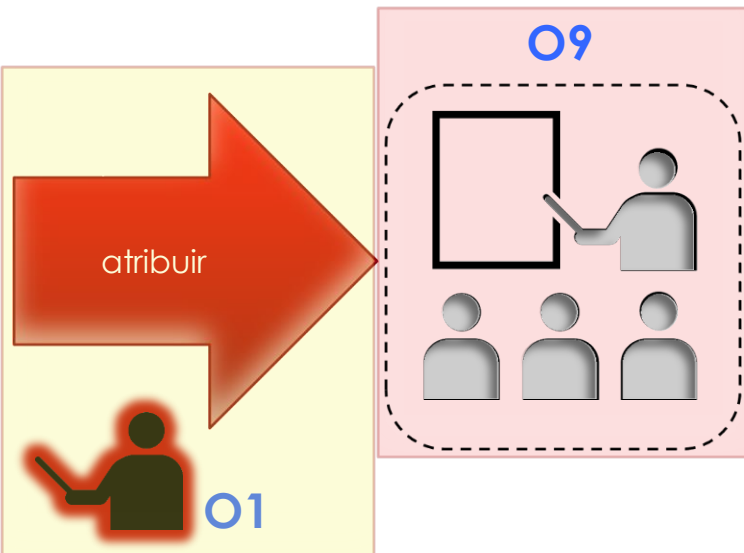


2.1. INVOCAÇÃO DE FUNCIONALIDADES

7

aula

RECETOR



MENSAGEM

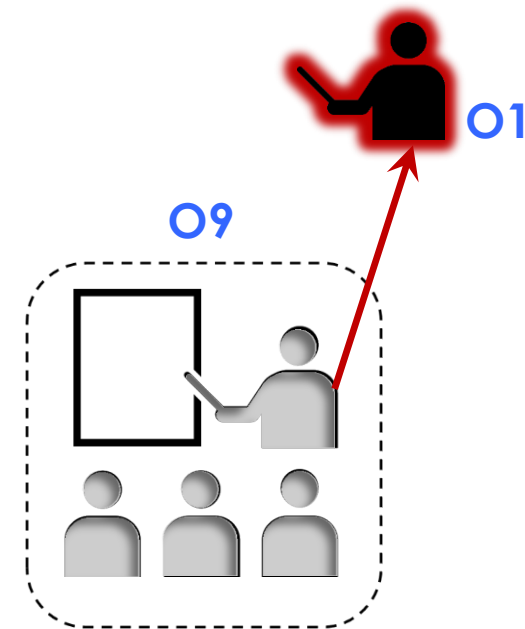
CONSTRUTOR

PROPRIEDADES

nome
número
sumário
professor
alunos

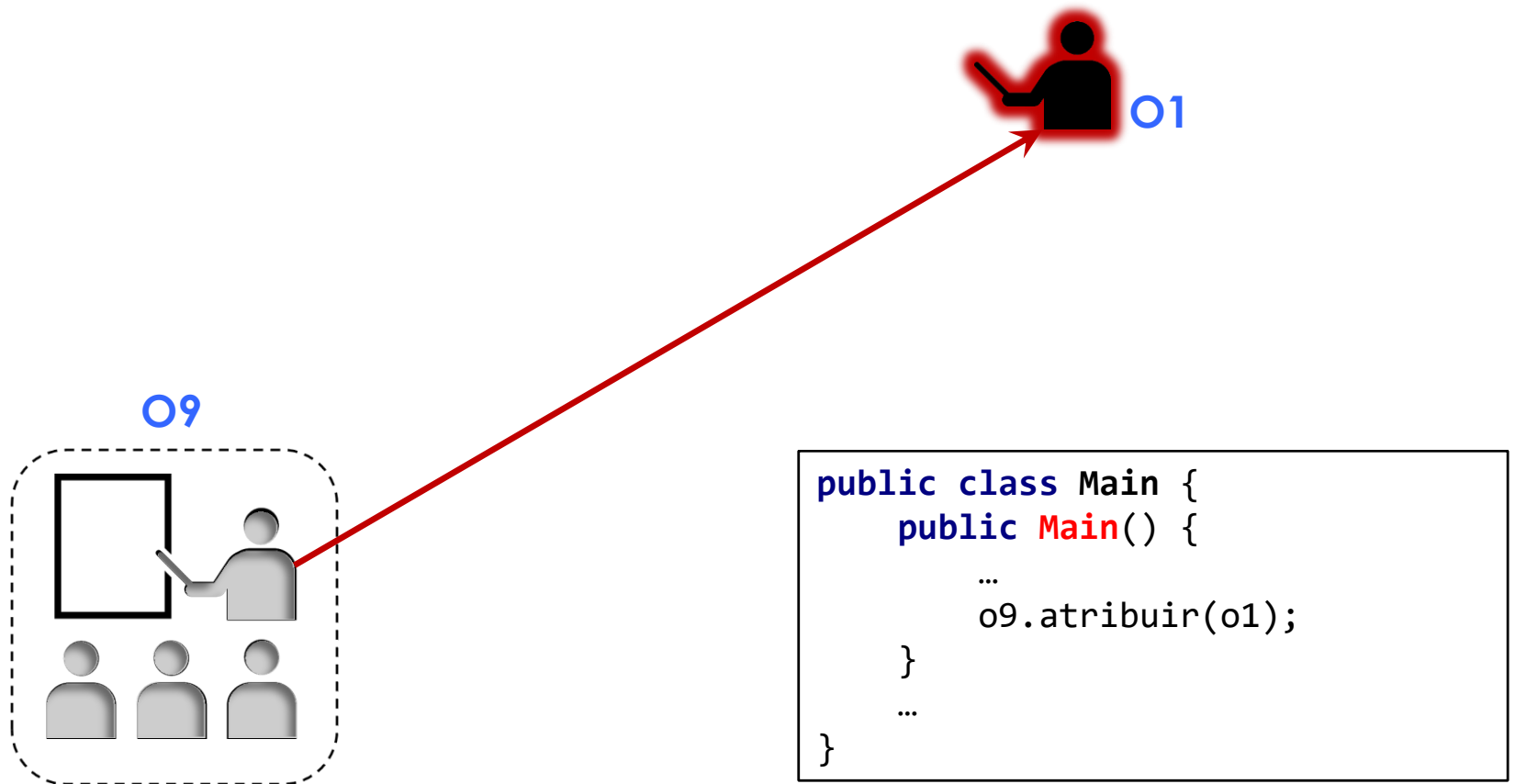
FUNCIONALIDADES

atribuir(professor)
adicionar(aluno)
adicionarLinhaSumário(linha)



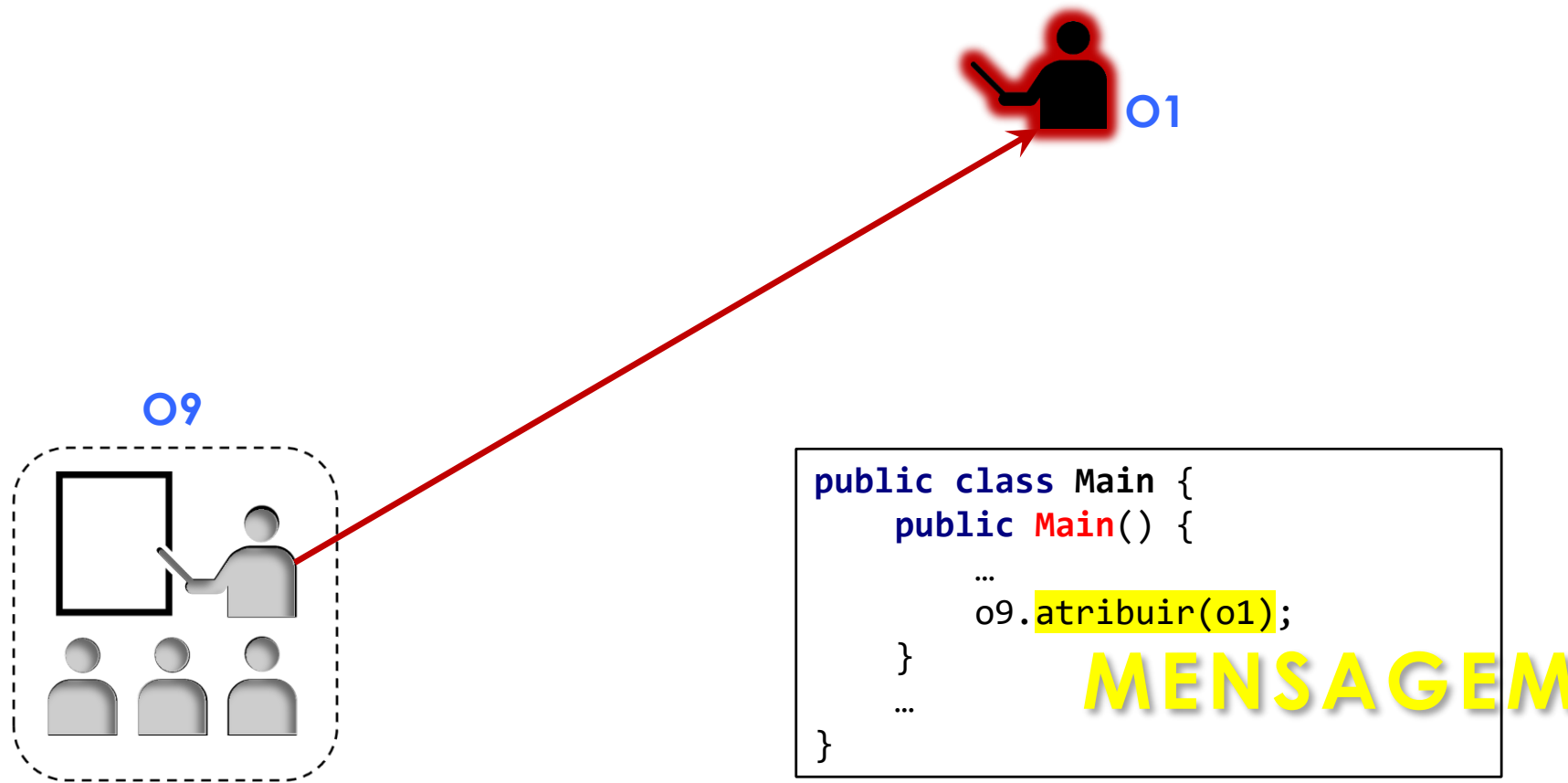
2.1. INVOCAÇÃO DE FUNCIONALIDADES

8



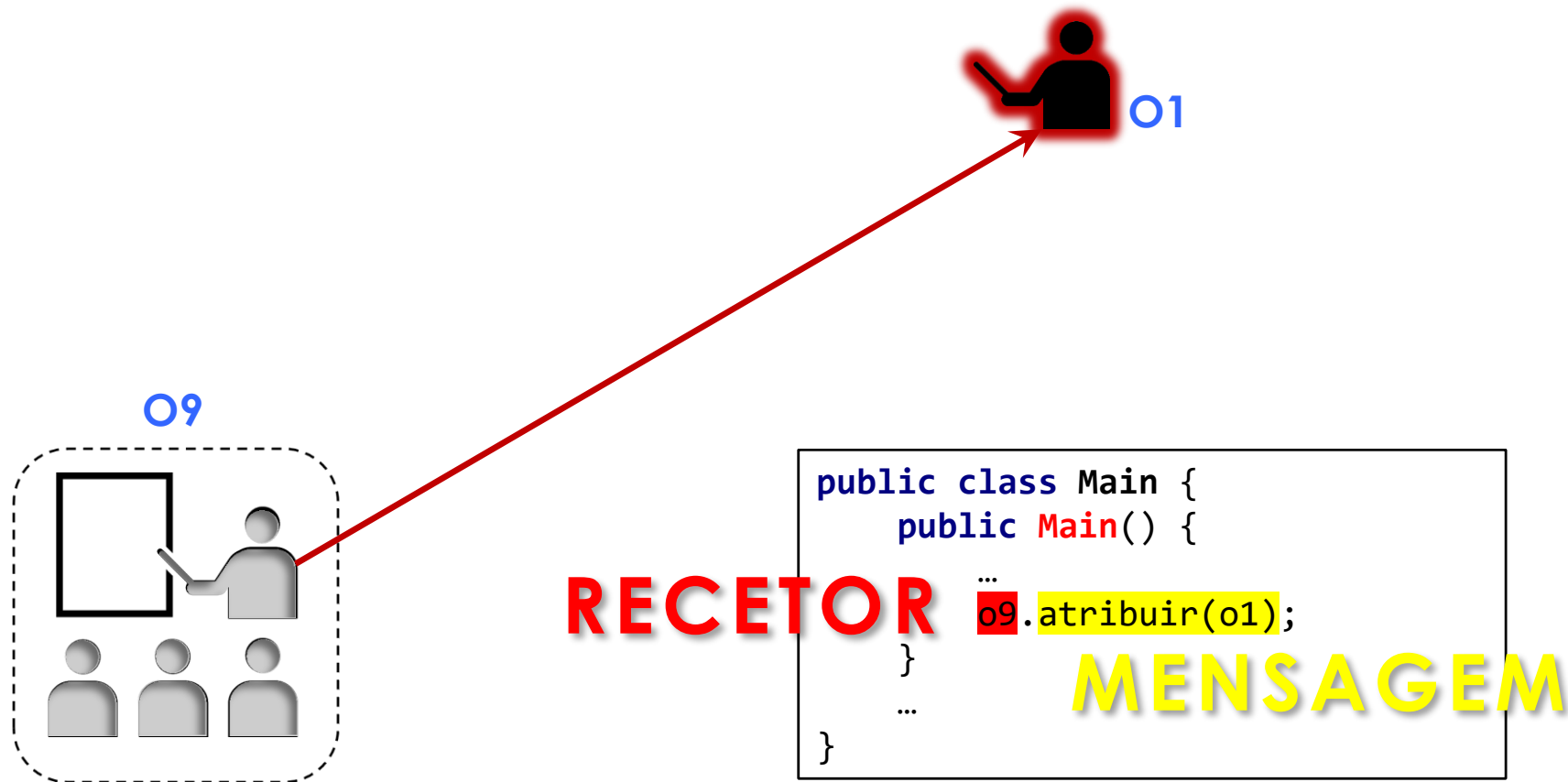
2.1. INVOCAÇÃO DE FUNCIONALIDADES

9



2.1. INVOCAÇÃO DE FUNCIONALIDADES

10



2.1. INVOCAÇÃO DE FUNCIONALIDADES

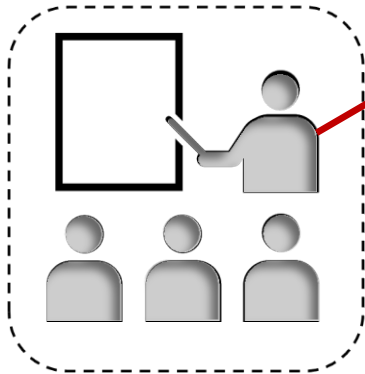
11

Aula

```
public void atribuir(Professor professor) {  
    this.professor = professor;  
}
```



o9

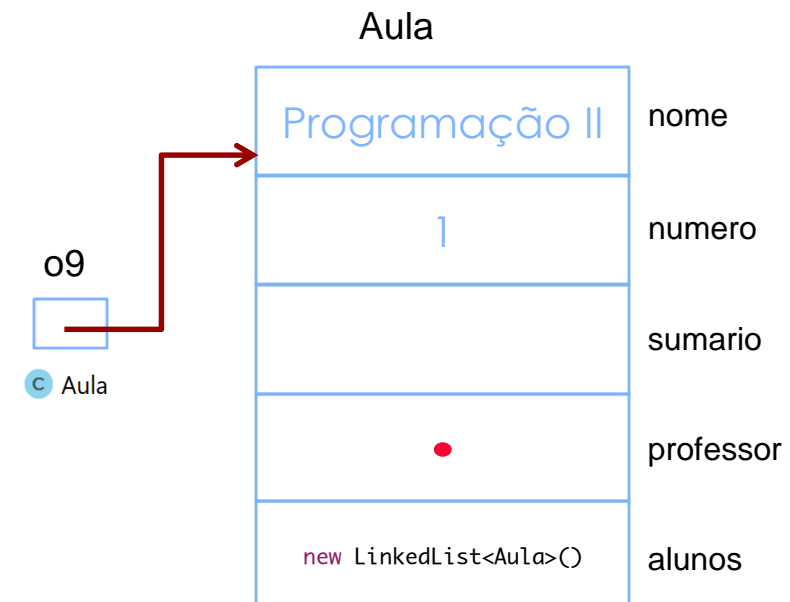
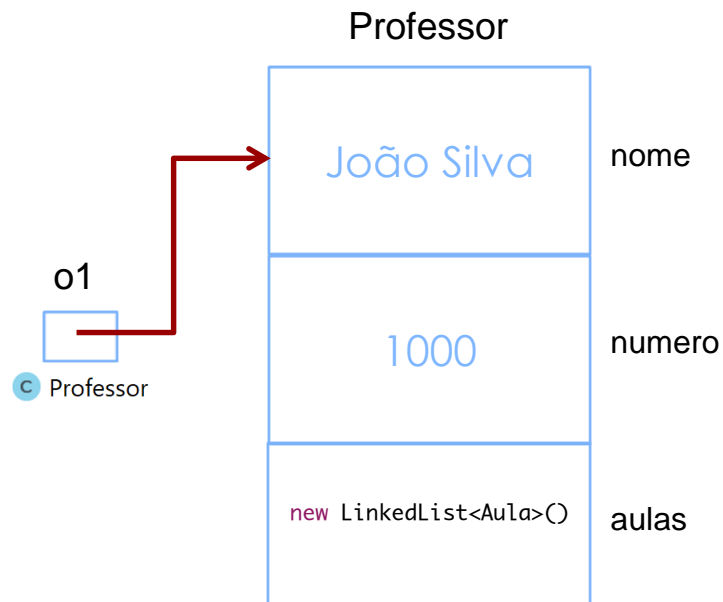


```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

2.1. INVOCAÇÃO DE FUNCIONALIDADES

12

```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```



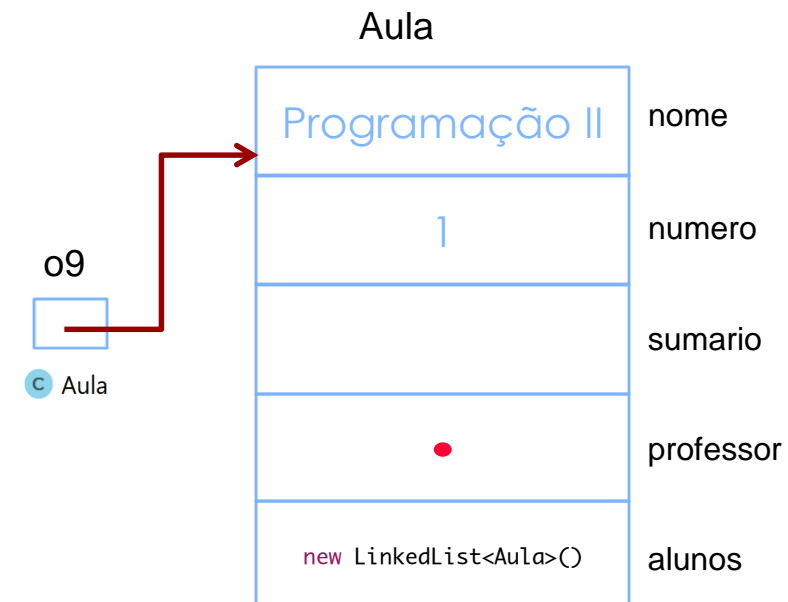
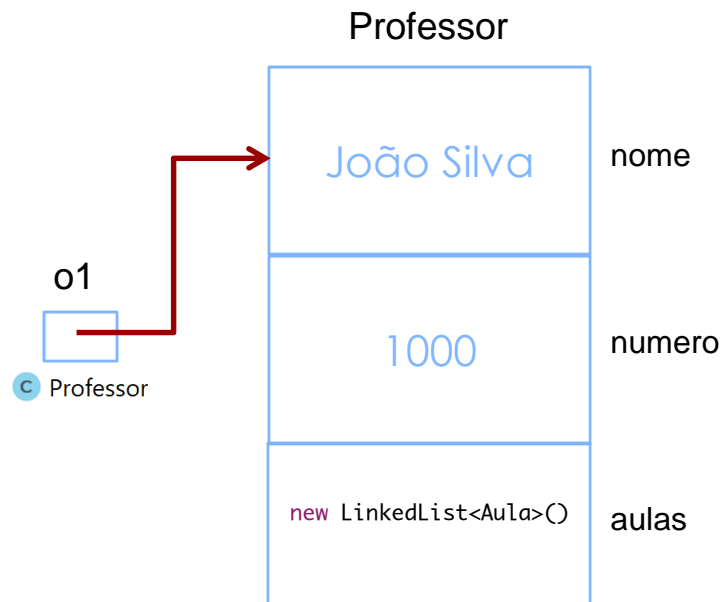
2.1. INVOCAÇÃO DE FUNCIONALIDADES

13

```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

internamente

```
atribuir(o9, o1);
```



2.1. INVOCAÇÃO DE FUNCIONALIDADES

14

```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

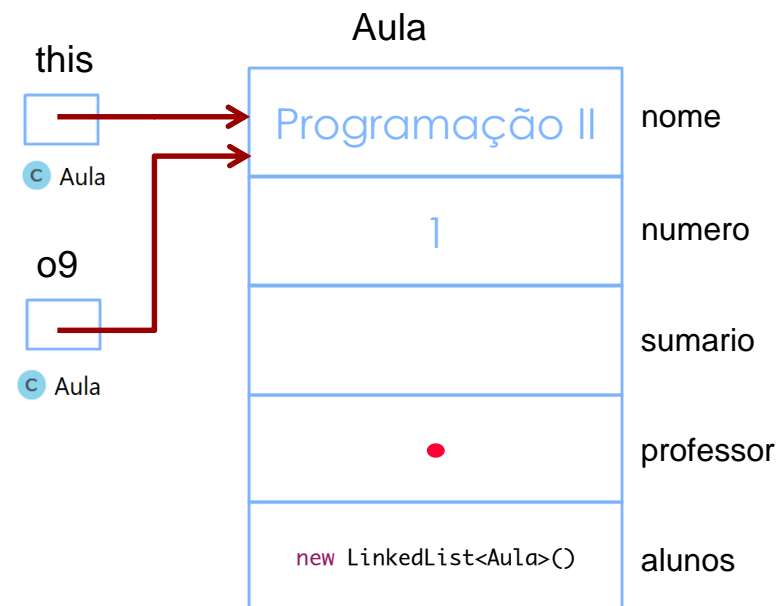
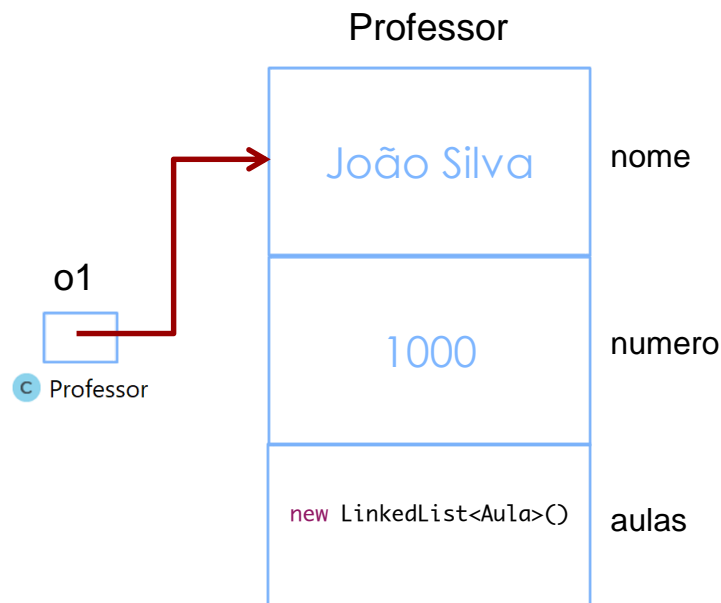
internamente

```
atribuir(o9, o1);
```

internamente

Aula

```
public void atribuir(Aula this, Professor professor) {  
    this.professor = professor;  
}
```



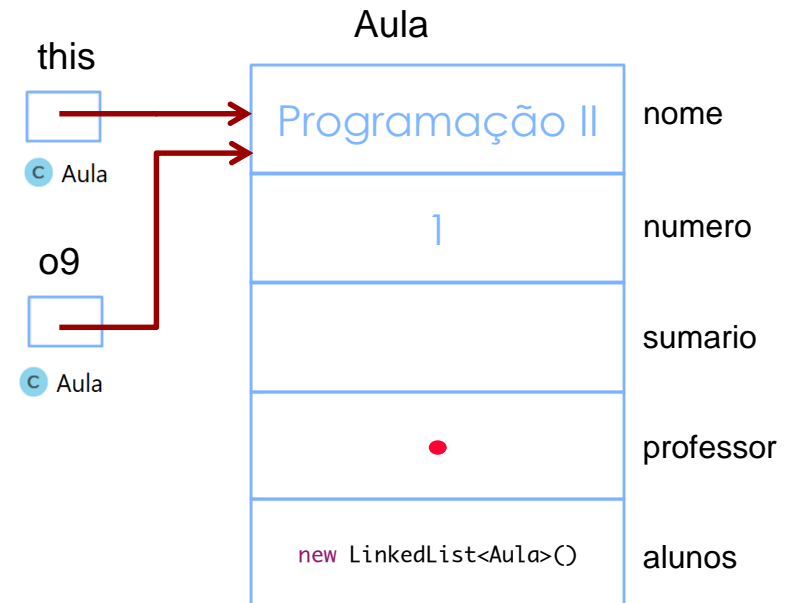
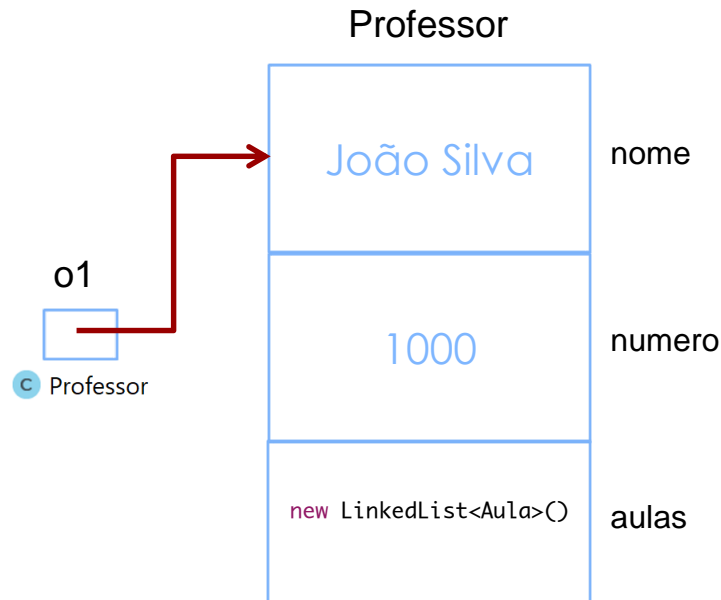
2.1. INVOCAÇÃO DE FUNCIONALIDADES

15

```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

Ⓢ Aula

```
public void atribuir(Professor professor) {  
    this.professor = professor;  
}
```



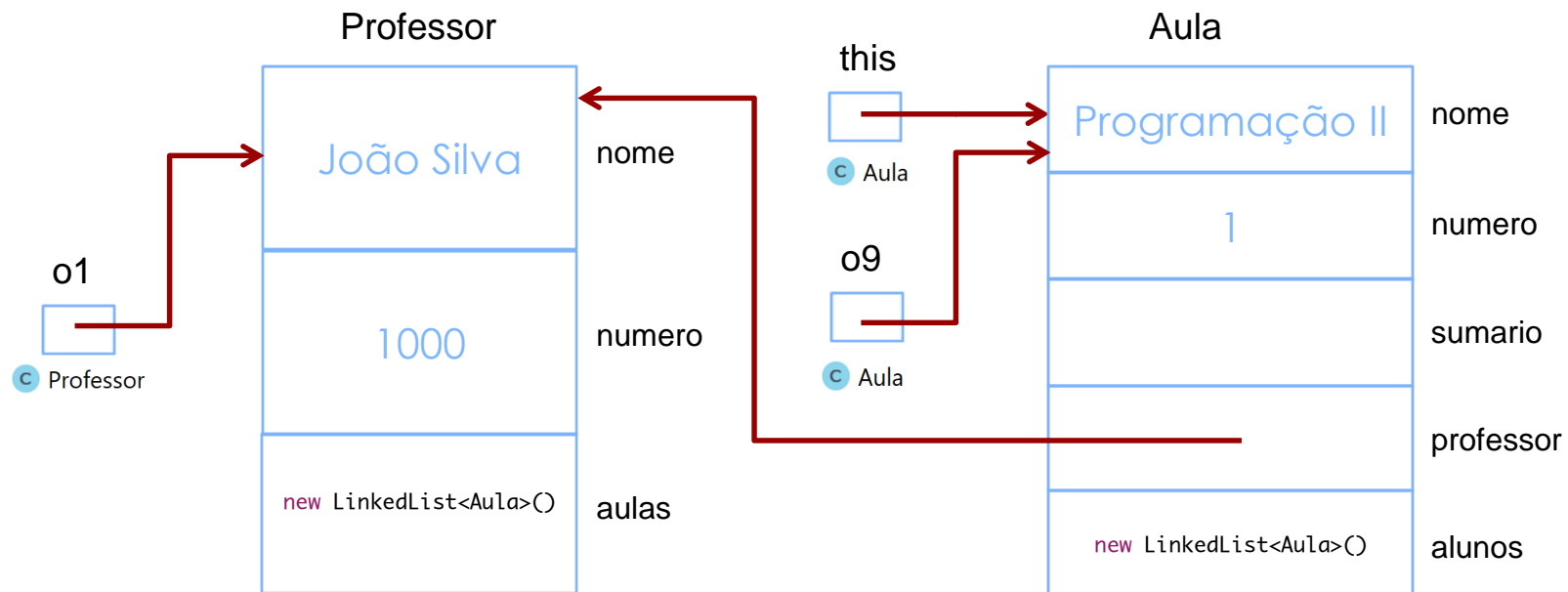
2.1. INVOCAÇÃO DE FUNCIONALIDADES

16

```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

Ⓢ Aula

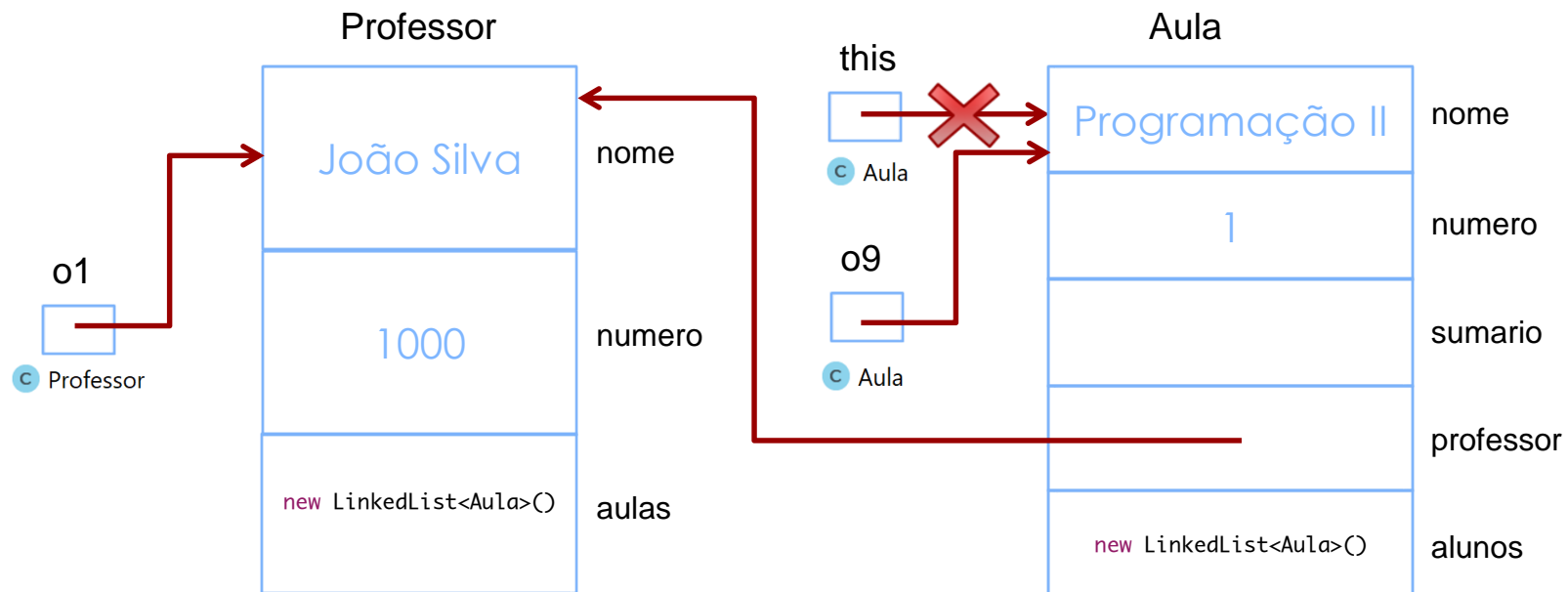
```
public void atribuir(Professor professor) {  
    this.professor = professor;  
}
```



2.1. INVOCAÇÃO DE FUNCIONALIDADES

17

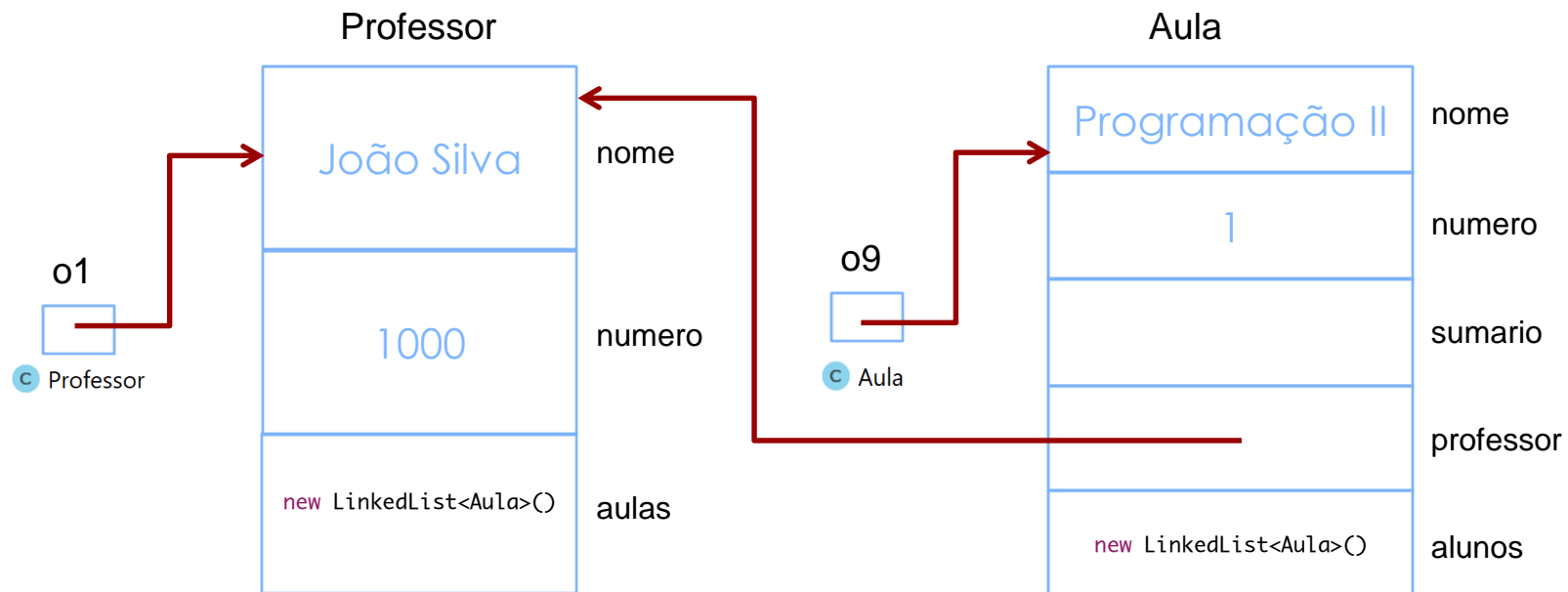
```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```



2.1. INVOCAÇÃO DE FUNCIONALIDADES

18

```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```



2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

- público

permite que a propriedade ou funcionalidade de um objeto possa ser acedida por código em qualquer parte do programa.

- protegido

a ser apresentado posteriormente (capítulo Herança).

- privado

determina que a propriedade ou funcionalidade só pode ser acedida por código dentro da própria classe.

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

20

PROPRIEDADES COM VISIBILIDADE PRIVADA

professor

aluno

aula

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
sumário
professor
alunos

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

atribuir(professor)
adicionar(aluno)
adicionarLinhaSumário(linha)

ENCAPSULAMENTO

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

21

PROPRIEDADES COM VISIBILIDADE PRIVADA

professor

aluno

aula

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
sumário
professor
alunos

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

atribuir(professor)
adicionar(aluno)
adicionarLinhaSumário(linha)

MÉTODOS COM

VISIBILIDADE PÚBLICA

ENCAPSULAMENTO

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

22

professor

aluno

aula

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
sumário
professor
alunos

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

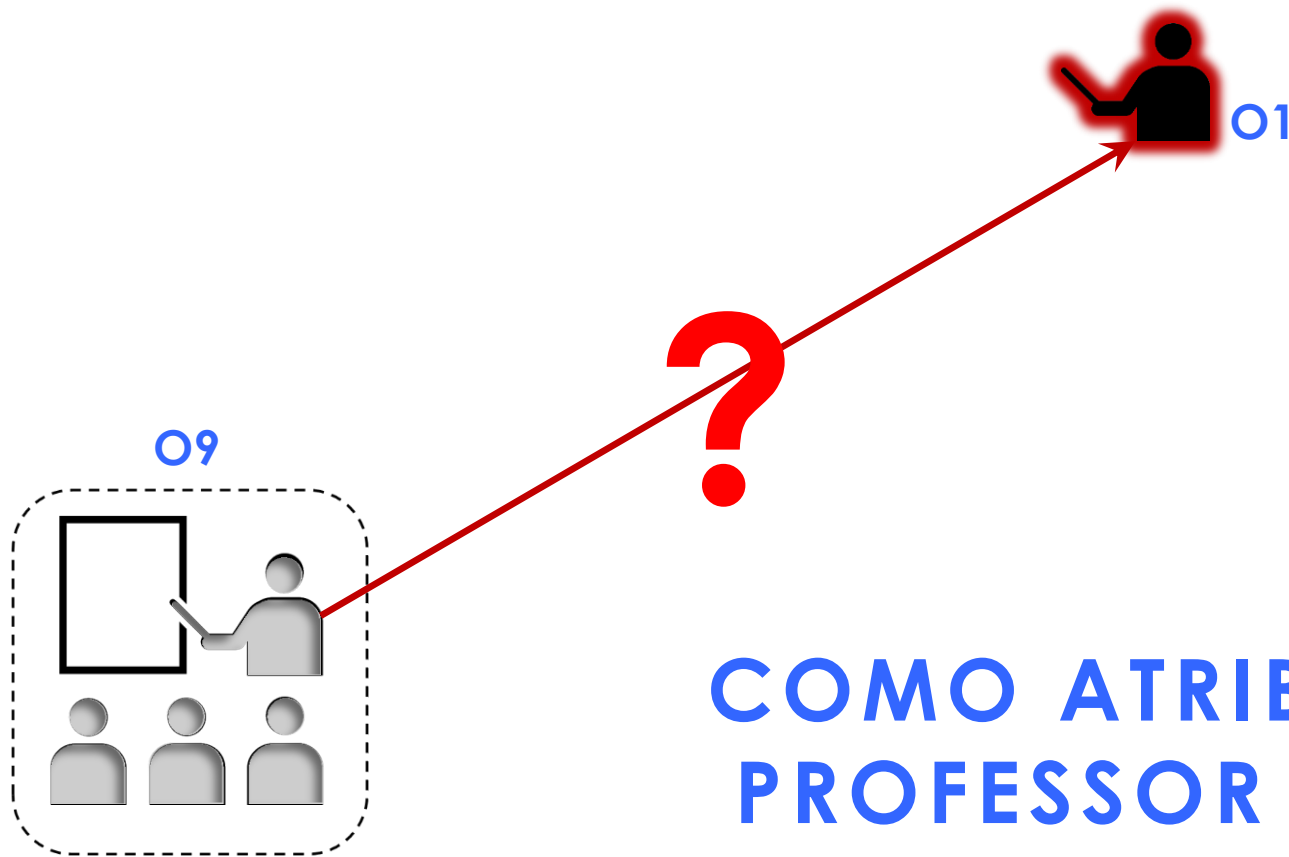
FUNCIONALIDADES

atribuir(professor)
adicionar(aluno)
adicionarLinhaSumário(linha)

PORQUÊ?

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

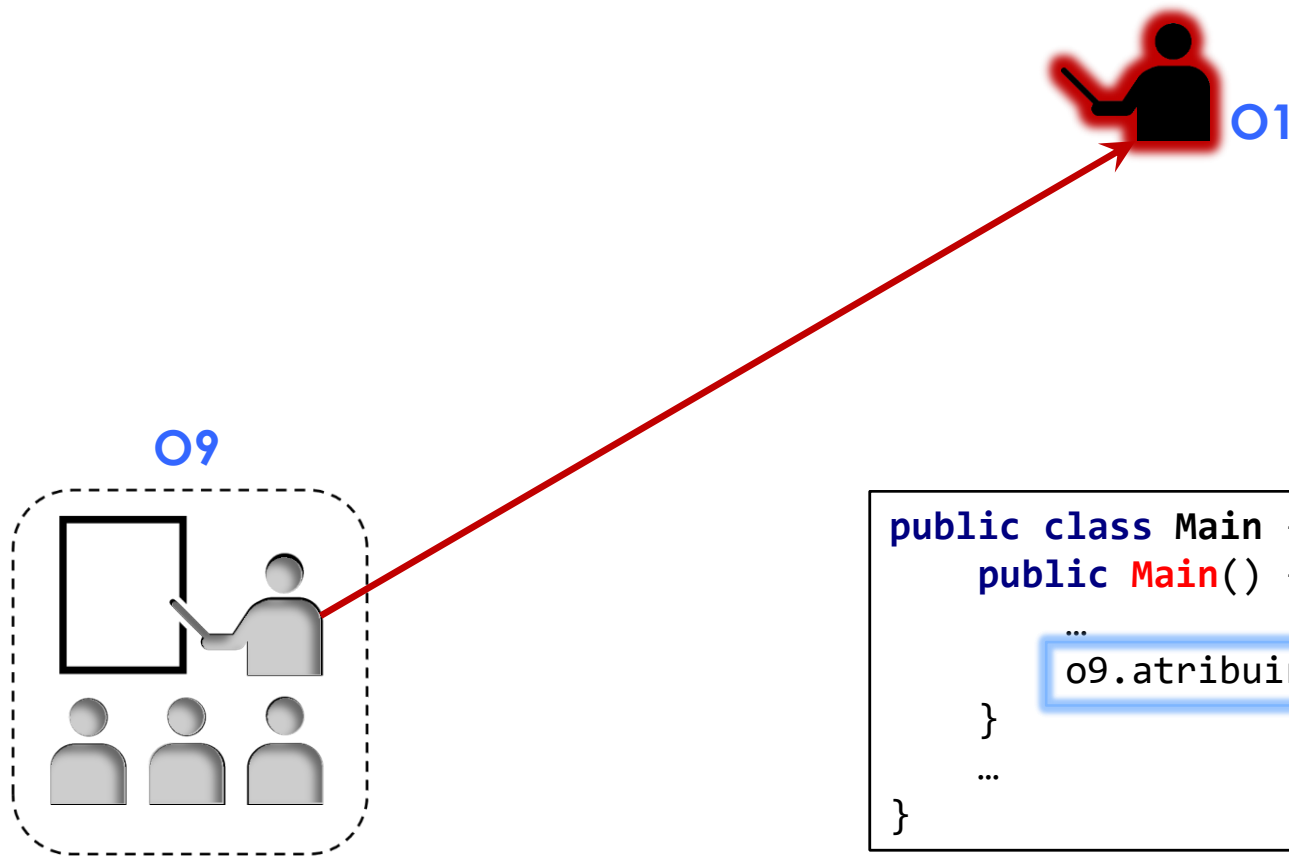
23



**COMO ATRIBUIR UM
PROFESSOR A UMA
AULA?**

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

24



SOLUÇÃO:

ATRIBUIÇÃO ATRAVÉS DA FUNCIONALIDADE atribuir(professor)

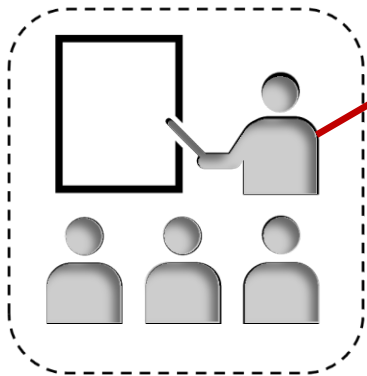
2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

25

Aula

```
public void atribuir(Professor professor) {  
    this.professor = professor;  
}
```

O9



```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

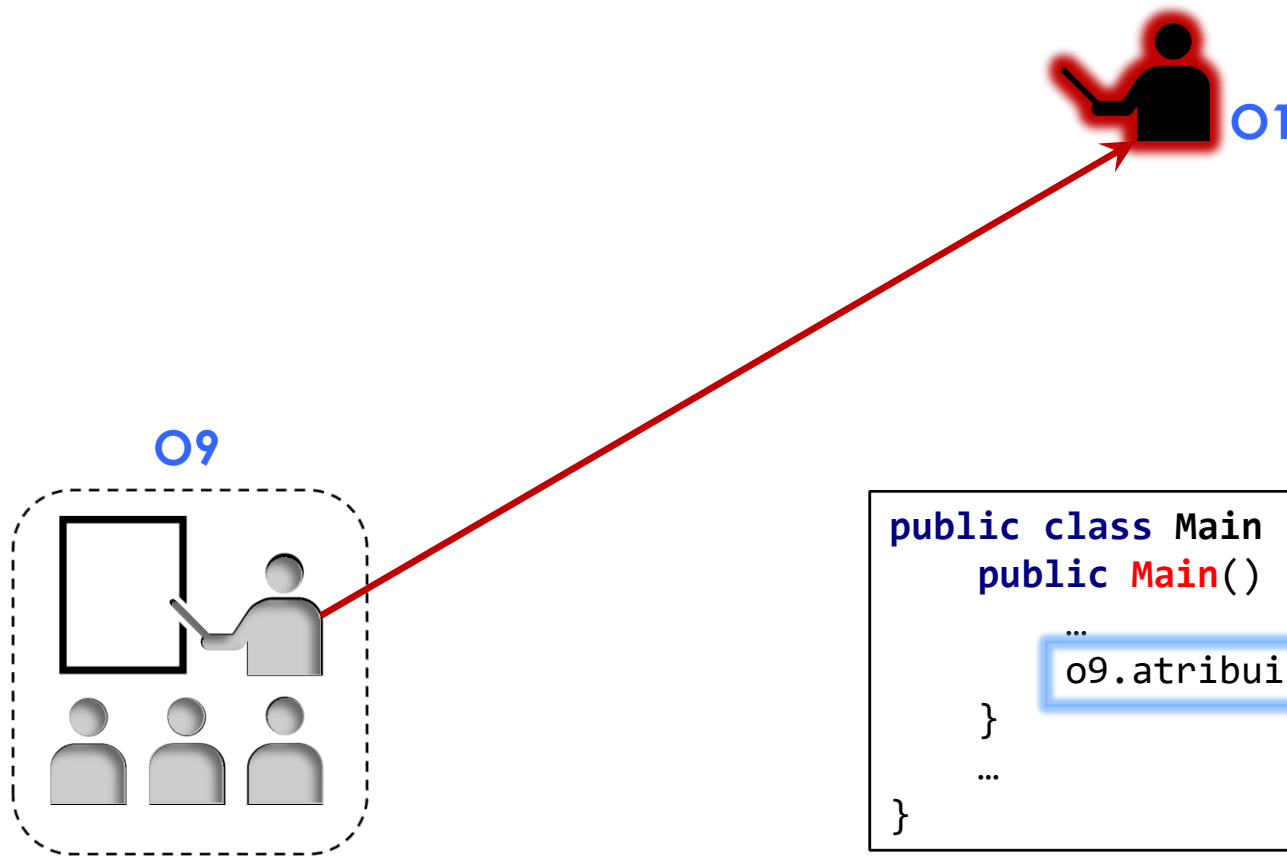


SOLUÇÃO:

ATRIBUIÇÃO ATRAVÉS DA FUNCIONALIDADE atribuir(professor)

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

26



```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```

SOLUÇÃO:

ATRIBUIÇÃO ATRAVÉS DA FUNCIONALIDADE atribuir(professor)

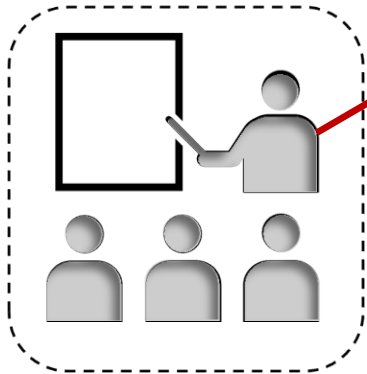
2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

27

Aula

```
private void atribuir(Professor professor) {  
    this.professor = professor;  
}
```

O9



```
public class Main {  
    public Main() {  
        ...  
        o9.atribuir(o1);  
    }  
    ...  
}
```



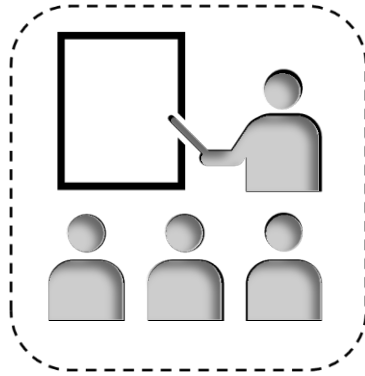
SOLUÇÃO:

ATRIBUIÇÃO ATRAVÉS DA FUNCIONALIDADE atribuir(professor)

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

28

09

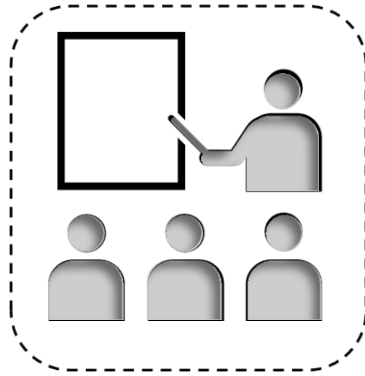


**COMO DEVE SER
PREENCHIDO O SUMÁRIO
DE UMA AULA?**

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

29

09



1ª. SOLUÇÃO

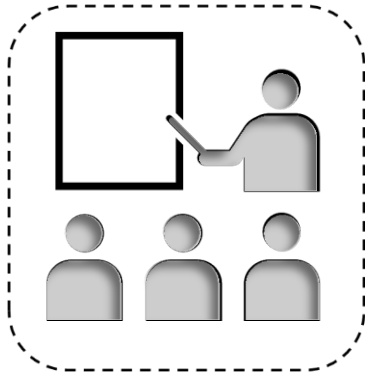
ACESSO PÚBLICO À PROPRIEDADE SUMÁRIO DA AULA POR PARTE DE QUALQUER OUTRO OBJETO

PERMITE SABER E ALTERAR O VALOR DESTA PROPRIEDADE

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

30

O9



```
public class Aula {  
    ...  
    public String sumario;  
    ...  
}
```

```
public class Main {  
    public Main() {  
        ...  
        o9.sumario = "Este é o sumário da aula.";  
    }  
    ...  
}
```

1ª. SOLUÇÃO

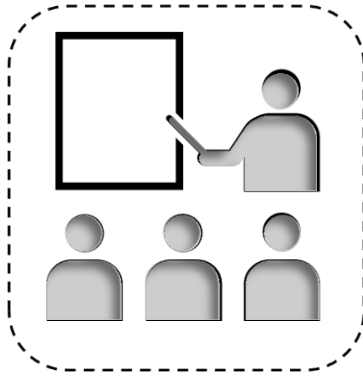
ACESSO PÚBLICO À PROPRIEDADE SUMÁRIO DA AULA POR PARTE DE QUALQUER OUTRO OBJETO

PERMITE SABER E ALTERAR O VALOR DESTA PROPRIEDADE

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

31

O9



```
public class Aula {  
    ...  
    public String sumario;  
    ...  
}
```

```
public class Main {  
    public Main() {  
        ...  
        o9.sumario = "Este é o sumário da aula.";  
    }  
    ...  
}
```

1ª. SOLUÇÃO

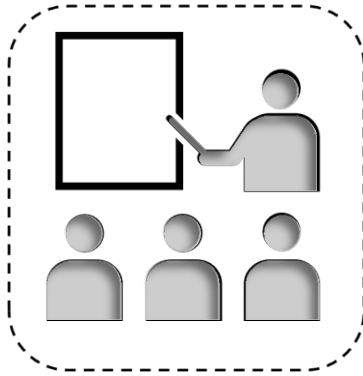
PROBLEMA!!!

O SUMÁRIO DA AULA O9 PODE SER PREENCHIDO SEM SER ATRAVÉS DA FUNCIONALIDADE `adicionarLinhaSumario(linha)`

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

32

O9



```
public class Aula {  
    ...  
    public String sumario;  
    ...  
}
```

```
public class Main {  
    public Main() {  
        ...  
        o9.sumario = "Este é o sumário da aula.";  
        o9.sumario = "Este é que é o  
                        VERDADEIRO!!!";  
    }  
    ...  
}
```

1ª. SOLUÇÃO

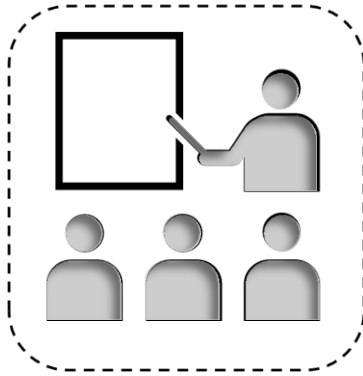
PROBLEMA!!!

O SUMÁRIO DA AULA O9 PODE SER PREENCHIDO
SEM SER ATRAVÉS DA FUNCIONALIDADE
adicionarLinhaSumario(linha)

2.2. MODIFICADORES DE ACESSO (VISIBILIDADE)

33

O9



```
public class Aula {  
    ...  
    private String sumario;  
    ...  
}
```

```
public class Main {  
    public Main() {  
        ...  
        o9.adicionarLinhaSumario("Assim se  
                                   preenche o sumário da aula.");  
    }  
    ...  
}
```

2ª. SOLUÇÃO

ACESSO PRIVADO À PROPRIEDADE SUMÁRIO DA AULA POR PARTE DE QUALQUER OUTRO OBJETO

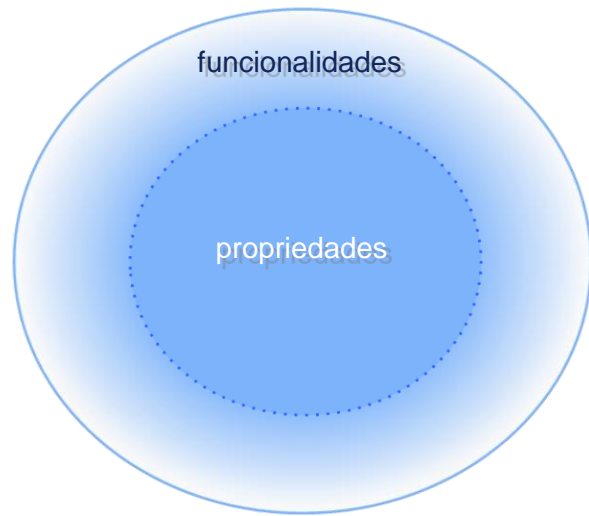
O SUMÁRIO DA AULA O9 APENAS PODE SER PREENCHIDO ATRAVÉS DA SUA FUNCIONALIDADE `adicionarLinhaSumario(linha)`

2.3. DEFINIÇÃO DE ENCAPSULAMENTO

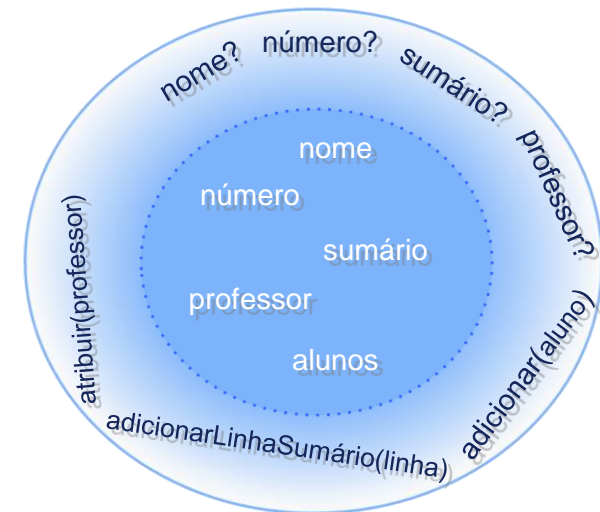
34

PROCESSO DE ENVOLVER PROPRIEDADES E FUNCIONALIDADES DE UMA CLASSE, PROTEGENDO A RESPECTIVA IMPLEMENTAÇÃO

classe



aula



2.3. DEFINIÇÃO DE ENCAPSULAMENTO

35

Em Java designam-se por acessores de uma classe os métodos públicos que permitem aceder e alterar o valor dos atributos

Por convenção em Java para cada atributo `a` do tipo `T` o método:

`T getA()` permite o acesso ao valor de `a`
`void setA(T t)` permite alterar o valor de `a`

2.4. EXEMPLO

36

professor

aluno

aula

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
aulas

PROPRIEDADES

nome
número
sumário
professor
alunos

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)

FUNCIONALIDADES

atribuir(professor)
adicionar(aluno)
adicionarLinhaSumário(linha)

CONSTRUTOR

CONSTRUTOR

CONSTRUTOR

2.4. EXEMPLO

37

professor



- O VALOR DO **número** PODE SER ACEDIDO E ALTERADO ATRAVÉS DOS SEUS ACESSORES **número?** E **atribuir(número)**
- O VALOR DO **nome** APENAS PODE SER ACEDIDO ATRAVÉS DO SEU ACESSOR **nome?**

NOTE QUE O PROFESSOR NÃO PERMITE ALTERAR O SEU **nome**, PELO QUE, NÃO FACULTA O ACESSOR **atribuir(nome)**

2.4. EXEMPLO

38

aluno

CONSTRUTOR

PROPRIEDADES

nome
número
aulas

FUNCIONALIDADES

adicionar(aula)
preencherSumário(aula)
número?
nome?
atribuir(número)

- O VALOR DO **número** PODE SER ACEDIDO E ALTERADO ATRAVÉS DOS SEUS ACESSORES **número?** E **atribuir(número)**
- O VALOR DO **nome** APENAS PODE SER ACEDIDO ATRAVÉS DO SEU ACESSOR **nome?**

2.4. EXEMPLO

39

aula

PROPRIEDADES

nome
número
sumário
professor
alunos

FUNCIONALIDADES

atribuir(professor)
adicionar(aluno)
adicionarLinhaSumário(linha
número?
nome?
atribuir(número)
sumário?
professor?

- O VALOR DO número PODE SER ACEDIDO E ALTERADO ATRAVÉS DOS SEUS ACESSORES número? E atribuir(número)
- O VALOR DO nome APENAS PODE SER ACEDIDO ATRAVÉS DO SEU ACESSOR nome?
- O VALOR DO sumário APENAS PODE SER ACEDIDO ATRAVÉS DO SEU ACESSOR sumário?
- O VALOR DO professor PODE SER ACEDIDO E ALTERADO ATRAVÉS DOS SEUS ACESSORES professor? E atribuir(professor)

2.4. EXEMPLO

40

	classes	Professor	Aluno	Aula
<i>tipo</i>	<i>atributo</i>			
String	nome	x	x	x
long	numero	x	x	x
LinkedList<Aula>	aulas	x	x	
String	sumario			x
Professor	professor			x
LinkedList<Aluno>	alunos			x
<i>return</i>	<i>método</i>			
void	setProfessor(Professor)			x
void	adicionar(Aula)	x	x	
void	preencherSumario(Aula)	x	x	
void	adicionar(Aluno)			x
void	adicionarLinhaSumario(String)			x
String	getNome()	x	x	x
long	getNumero()	x	x	x
void	setNumero(long)	x	x	x
String	getSumario()			x
Professor	getProfessor()			x

TABELA COM
CARACTERÍSTICAS
DAS CLASSES

2.4. EXEMPLO

41

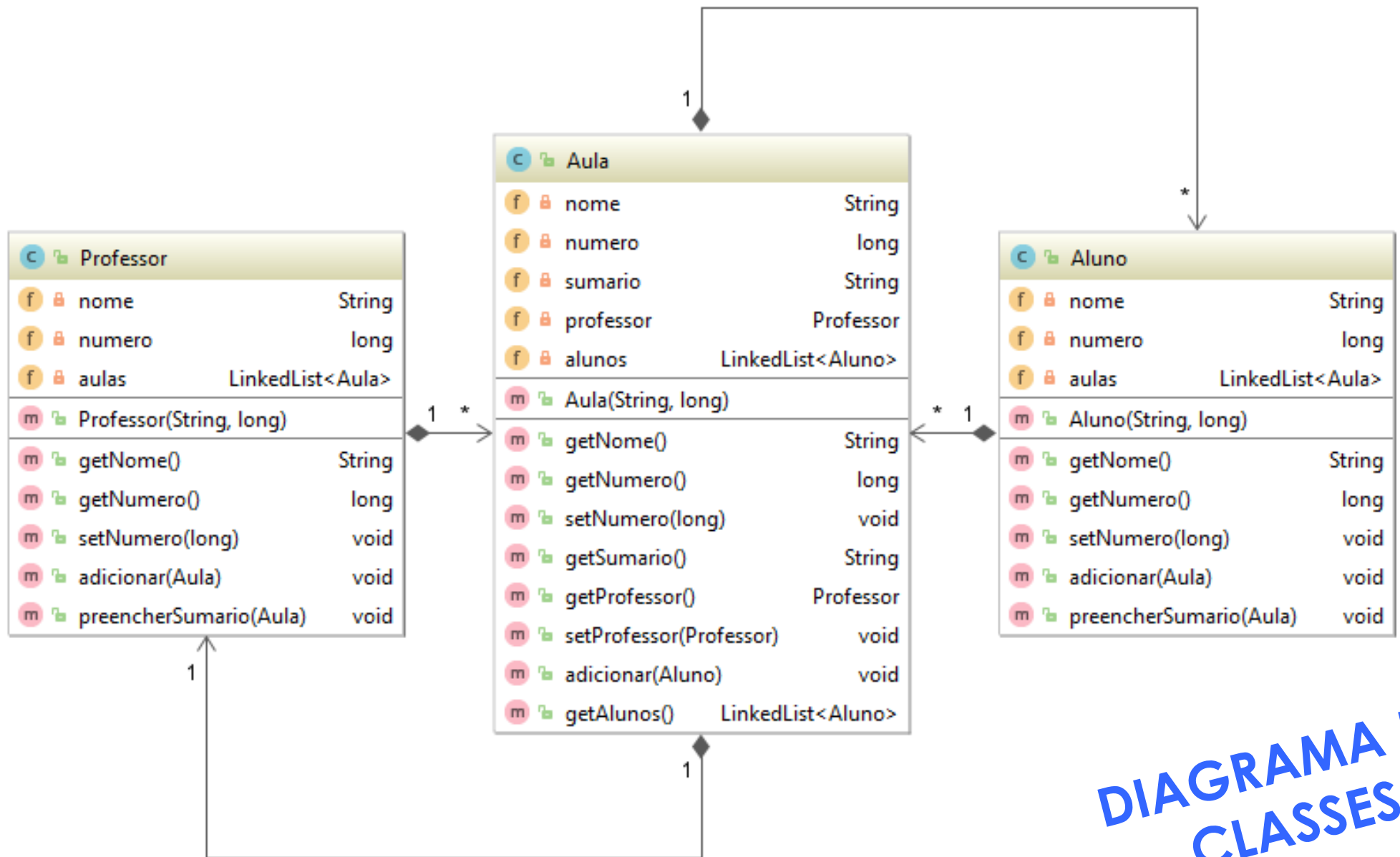


DIAGRAMA DE
CLASSES

2.4. EXEMPLO

42

```
public class Professor {  
  
    private String nome;  
    private long numero;  
    private LinkedList<Aula> aulas;
```

...

```
public String getNome() {  
    return nome;  
}  
  
public long getNumero() {  
    return numero;  
}  
  
public void setNumero(long numero) {  
    this.numero = numero;  
}
```

...

```
}
```

GETTER
SETTER

2.4. EXEMPLO

43

```
public class Professor {  
  
    private String nome;  
    private long numero;  
    private LinkedList<Aula> aulas;  
  
    ...  
  
    public String getNome() {  
        return nome;  
    }  
  
    public long getNumero() {  
        return numero;  
    }  
  
    public void setNumero(long numero) {  
        this.numero = numero;  
    }  
  
    ...  
}
```

SEMPRE QUE NÃO
SEJA NECESSÁRIO
O JAVA PERMITE O
ACESSO IMPLÍCITO
AOS ATRIBUTOS E
MÉTODOS DENTRO
DA CLASSE

2.4. EXEMPLO

44

```
public class Aluno {  
  
    private String nome;  
    private long numero;  
    private LinkedList<Aula> aulas;
```

...

```
public String getNome() {  
    return nome;  
}  
  
public long getNumero() {  
    return numero;  
}  
  
public void setNumero(long numero) {  
    this.numero = numero;  
}
```

...

```
}
```

GETTER
SETTER

2.4. EXEMPLO

45

```
public class Aula {  
    private String nome;  
    private long numero;  
    private String sumario;  
    private Professor professor;  
    private LinkedList<Aluno> alunos;
```

...

```
public String getNome() {  
    return nome;  
}  
  
public long getNumero() {  
    return numero;  
}  
  
public void setNumero(long numero) {  
    this.numero = numero;  
}
```

...

GETTER
SETTER

2.4. EXEMPLO

46

...

```
public String getSumario() {  
    return sumario;  
}  
  
public Professor getProfessor() {  
    return professor;  
}  
  
public void setProfessor(Professor professor) {  
    this.professor = professor;  
}
```

GETTER
SETTER

```
}
```

2.4. EXEMPLO

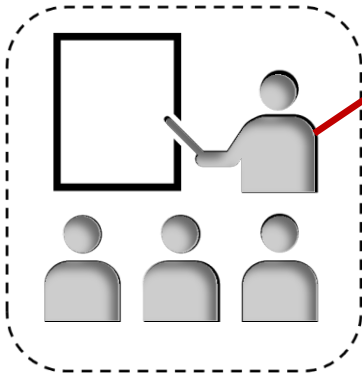
47

Aula

```
public void setProfessor(Professor professor) {  
    this.professor = professor;  
}
```



O9



Main

```
public Main() {  
    ...  
    o9.setProfessor(o1);  
}
```

2.4. EXEMPLO

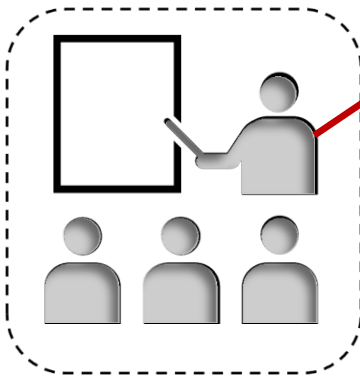
48

Aula

```
public void setProfessor(Professor professor) {  
    this.professor = professor;  
}
```



o9



Main

```
public Main() {  
    ...  
    o9.setProfessor(o1);  
}
```

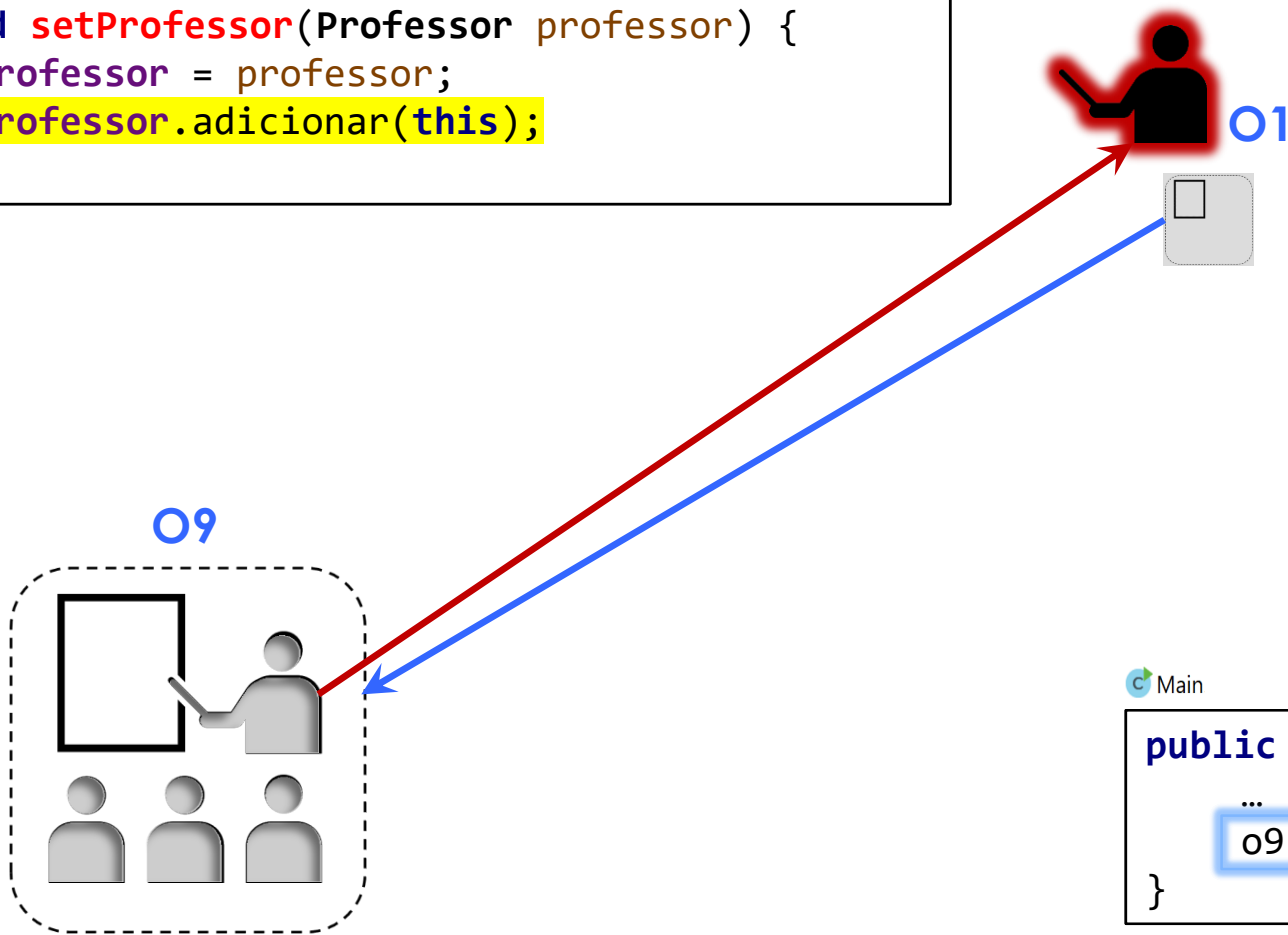
AO ASSOCIAR UM PROFESSOR A UMA
AULA NÃO DEVERÍAMOS ADICIONAR
ESSA AULA A ESSE PROFESSOR?

2.4. EXEMPLO

49

Aula

```
public void setProfessor(Professor professor) {  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

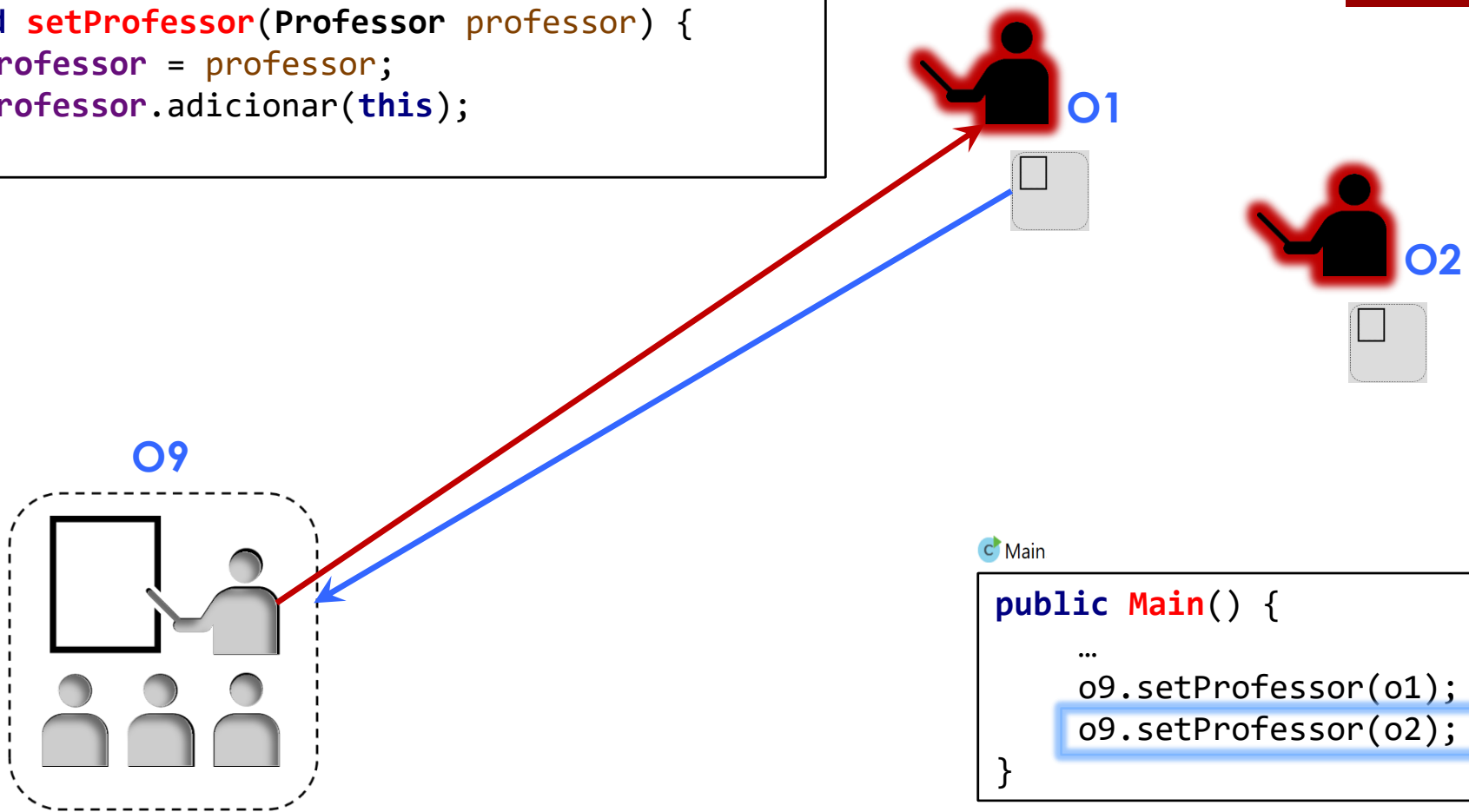
```
public Main() {  
    ...  
    o9.setProfessor(o1);  
}
```

2.4. EXEMPLO

50

Aula

```
public void setProfessor(Professor professor) {  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

```
public Main() {  
    ...  
    o9.setProfessor(o1);  
    o9.setProfessor(o2);  
}
```

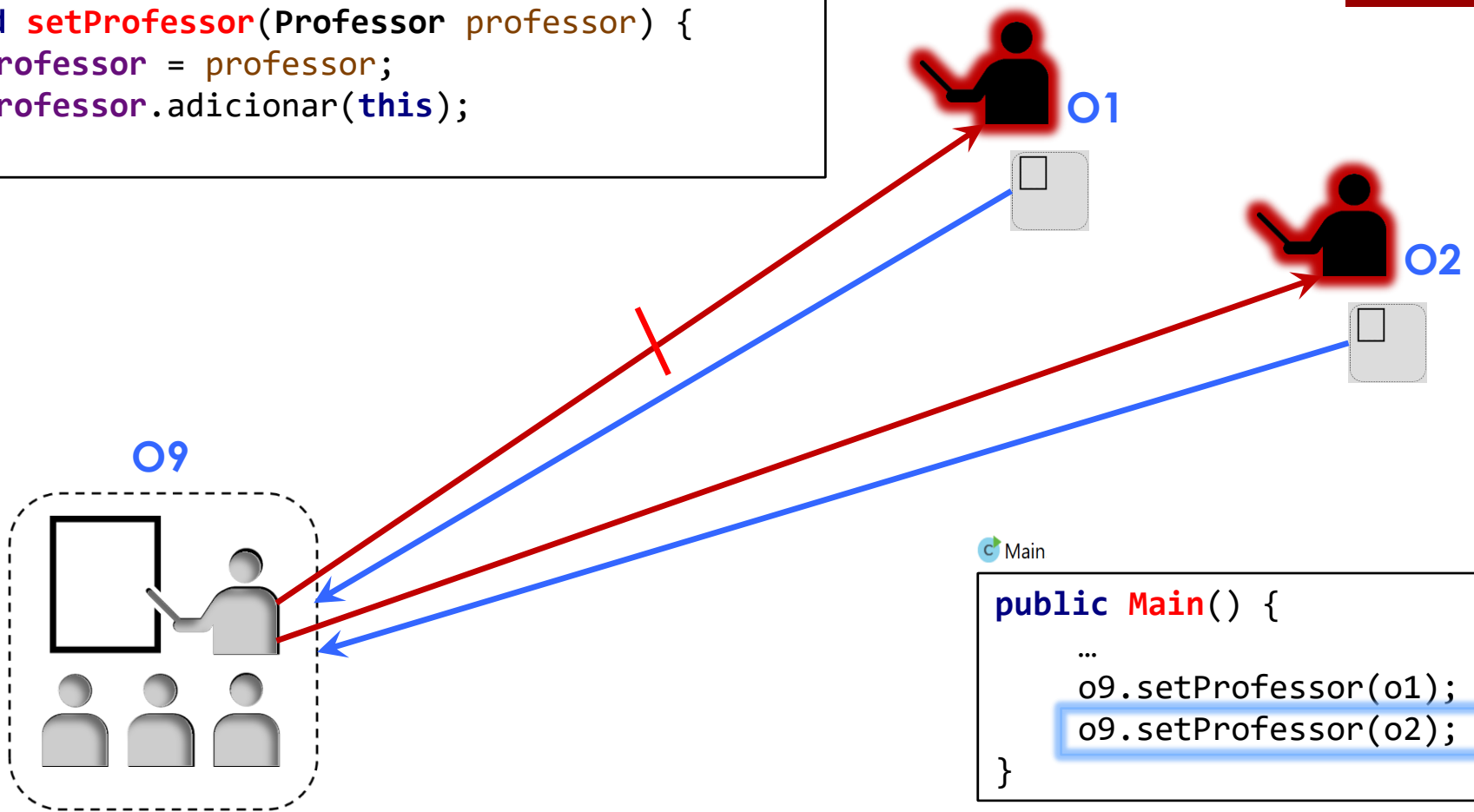
E O QUE ACONTECE AO ASSOCIARMOS
O PROFESSOR O2 À AULA O9?

2.4. EXEMPLO

51

Aula

```
public void setProfessor(Professor professor) {  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```

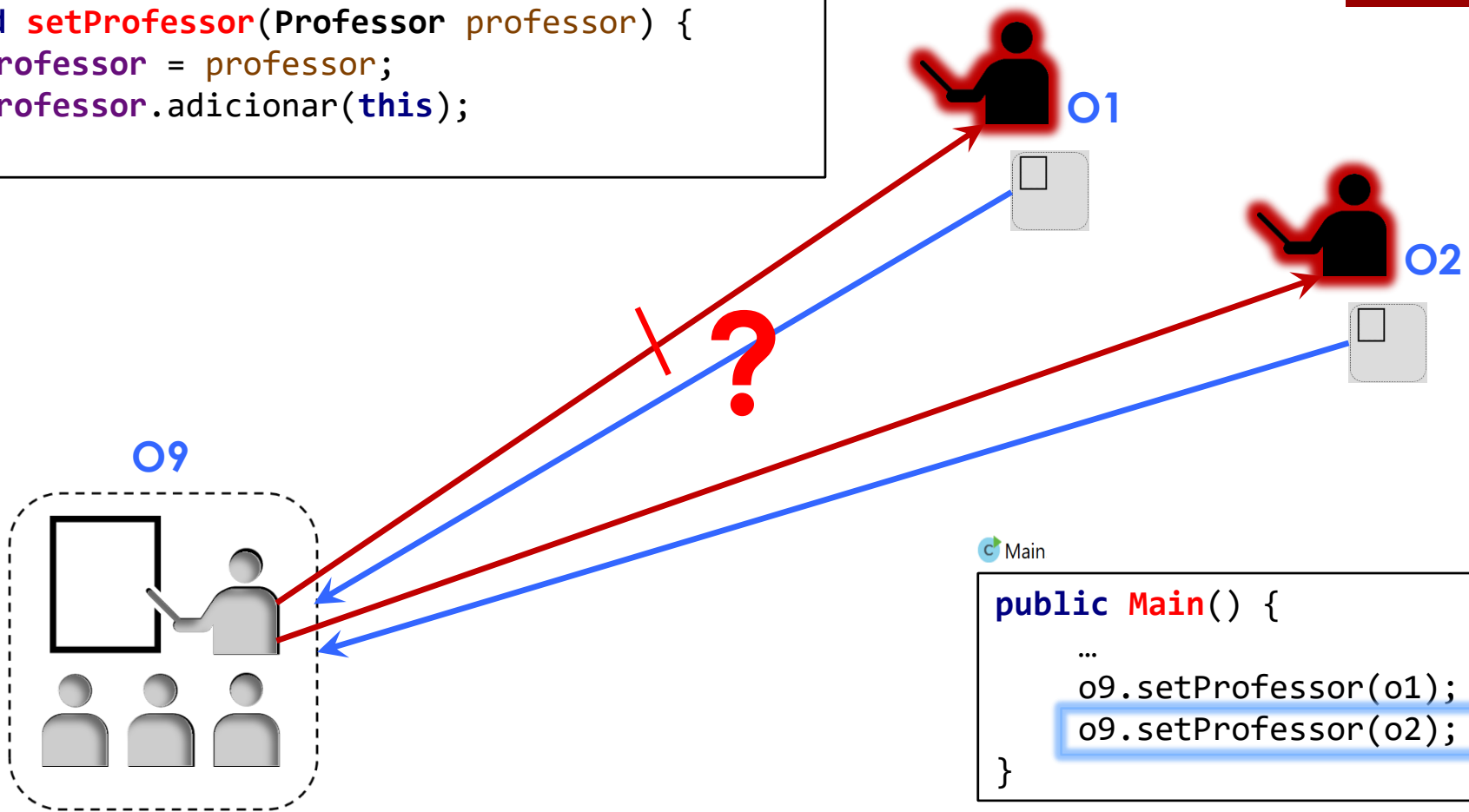


2.4. EXEMPLO

52

Aula

```
public void setProfessor(Professor professor) {  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

```
public Main() {  
    ...  
    o9.setProfessor(o1);  
    o9.setProfessor(o2);  
}
```

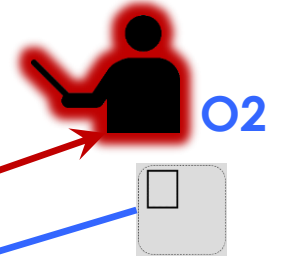
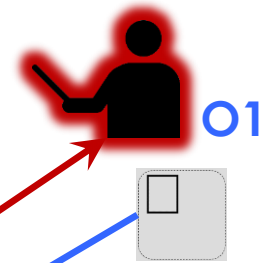
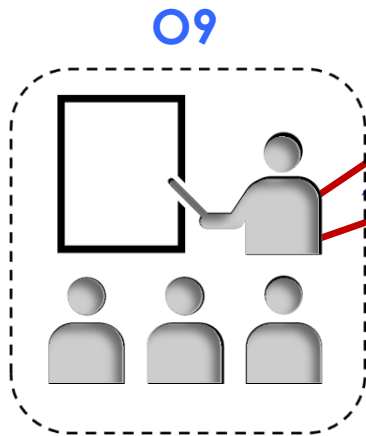
E O QUE ACONTECE ÀS AULAS DO
PROFESSOR O1 AO ASSOCIARMOS O
PROFESSOR O2 À AULA O9?

2.4. EXEMPLO

53

Aula

```
public void setProfessor(Professor professor) {  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

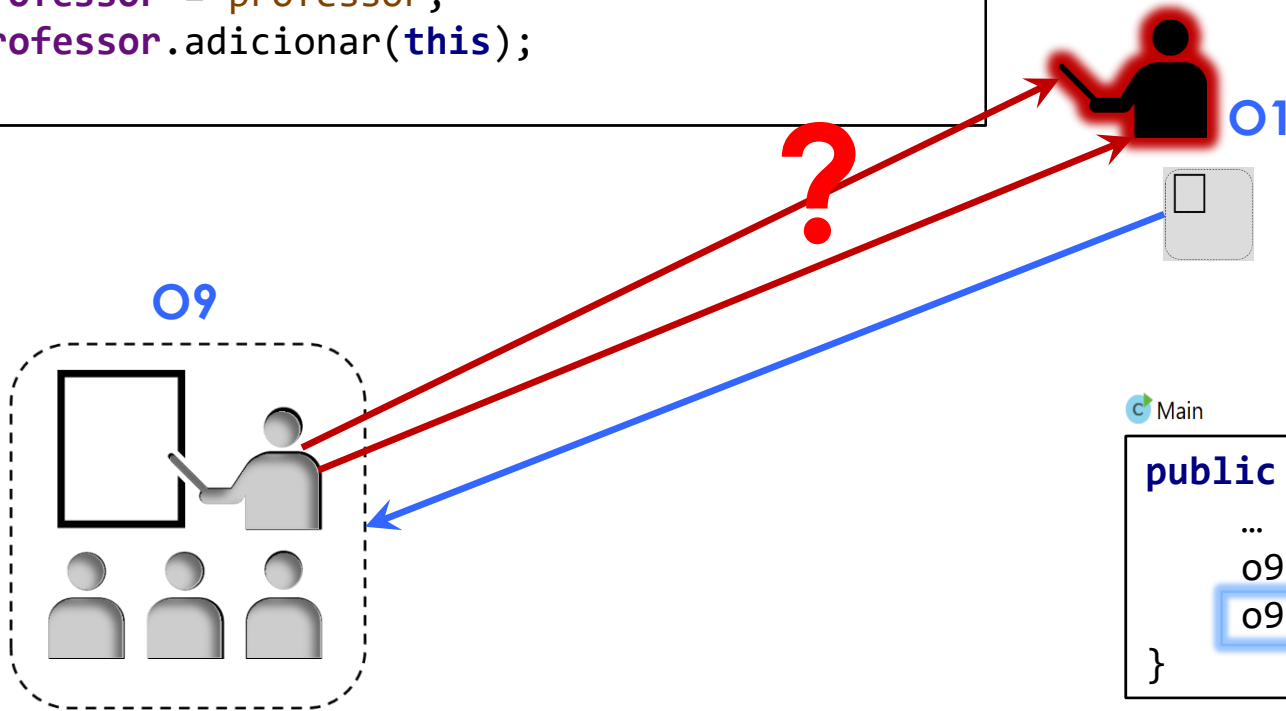
```
public Main() {  
    ...  
    o9.setProfessor(o1);  
    o9.setProfessor(o2);  
}
```

2.4. EXEMPLO

54

Aula

```
public void setProfessor(Professor professor) {  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

```
public Main() {  
    ...  
    o9.setProfessor(o1);  
    o9.setProfessor(o1);  
}
```

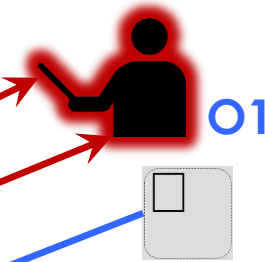
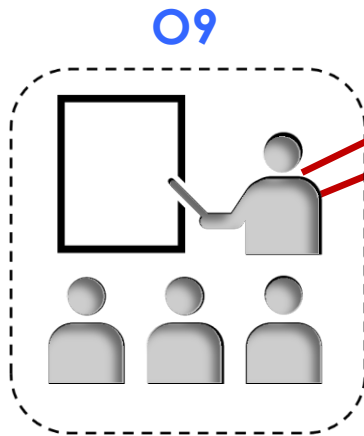
**E O QUE ACONTECE SE
ASSOCIARMOS A UMA AULA O
PROFESSOR JÁ ASSOCIADO?**

2.4. EXEMPLO

55

Aula

```
public void setProfessor(Professor professor) {  
    if (this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

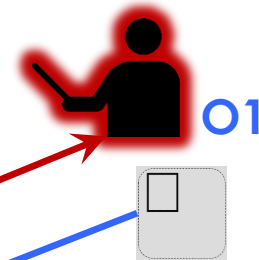
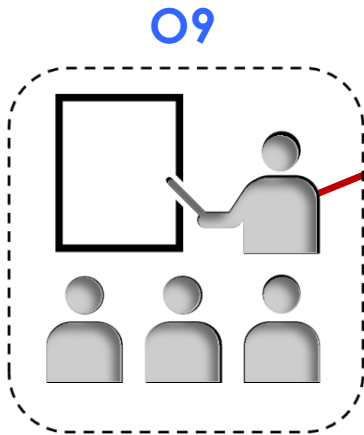
```
public Main() {  
    ...  
    o9.setProfessor(o1);  
    o9.setProfessor(o1);  
}
```

2.4. EXEMPLO

56

Aula

```
public void setProfessor(Professor professor) {  
    if (this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



Main

```
public Main() {  
    ...  
    o9.setProfessor(o1);  
    o9.setProfessor(o1);  
}
```


2.4. EXEMPLO

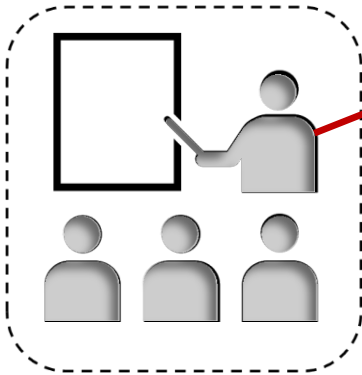
57

Aula

```
public void setProfessor(Professor professor) {  
    if (this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



O9



Main

```
public Main() {  
    ...  
    o9.setProfessor(null);  
}
```

**E O QUE ACONTECE SE ASSOCIARMOS
UMA AULA A UM PROFESSOR ATRAVÉS
DE UMA REFERÊNCIA NULA?**

2.4. EXEMPLO

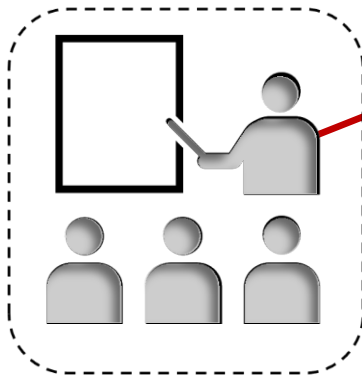
58

Aula

```
public void setProfessor(Professor professor) {  
    if (professor == null || this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



O9



Main

```
public Main() {  
    ...  
    o9.setProfessor(null);  
}
```

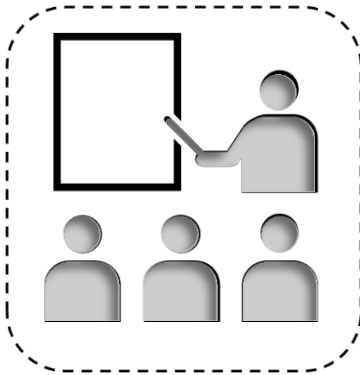
2.4. EXEMPLO

59

Aula

```
public void setProfessor(Professor professor) {  
    if (professor == null || this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```

O9

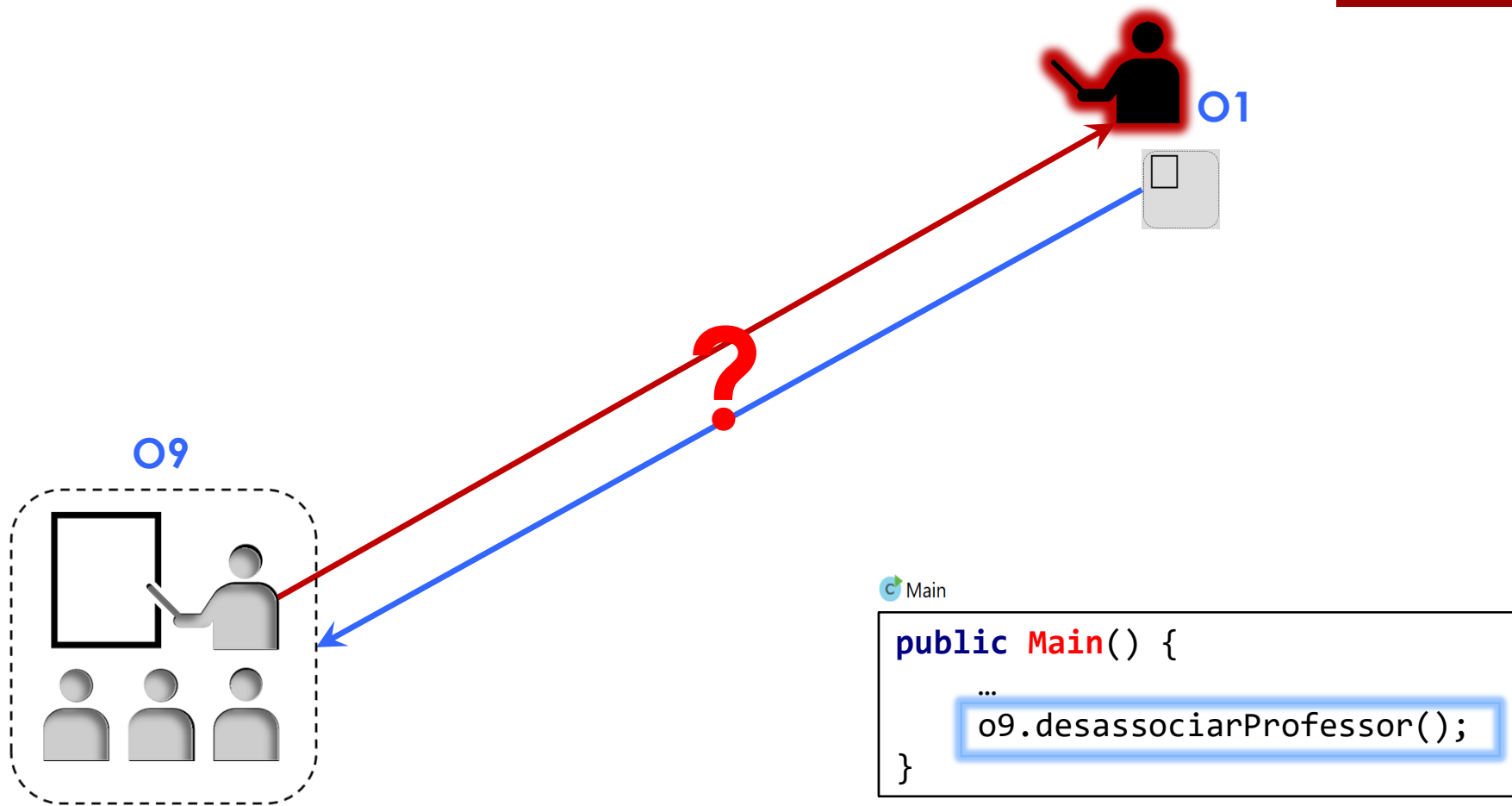


Main

```
public Main() {  
    ...  
    o9.setProfessor(null);  
}
```

2.4. EXEMPLO

60



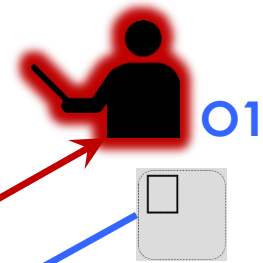
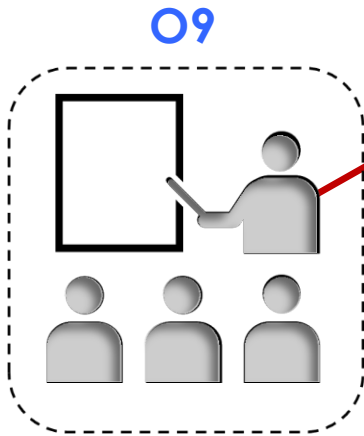
**E COMO DESASSOCIAR UM
PROFESSOR DE UMA AULA?**

2.4. EXEMPLO

61

Aula

```
public void desassociarProfessor() {  
    if (professor == null) {  
        return;  
    }  
    Professor professorARemover = professor;  
    professor = null;  
    professorARemover.remover(this);  
}
```



Main

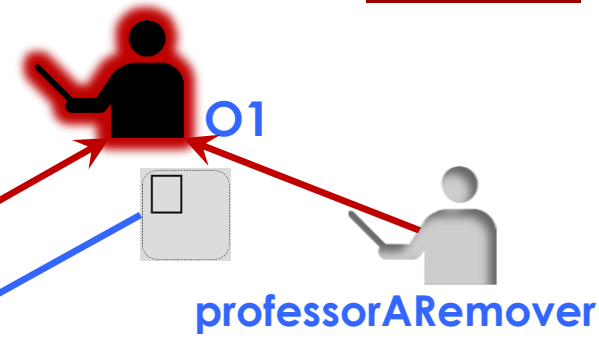
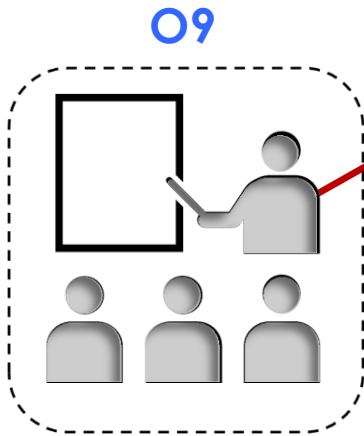
```
public Main() {  
    ...  
    o9.desassociarProfessor();  
}
```

2.4. EXEMPLO

62

Aula

```
public void desassociarProfessor() {  
    if (professor == null) {  
        return;  
    }  
    Professor professorARemover = professor;  
    professor = null;  
    professorARemover.remover(this);  
}
```



Main

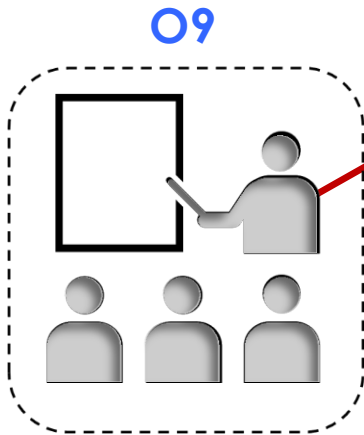
```
public Main() {  
    ...  
    o9.desassociarProfessor();  
}
```

2.4. EXEMPLO

63

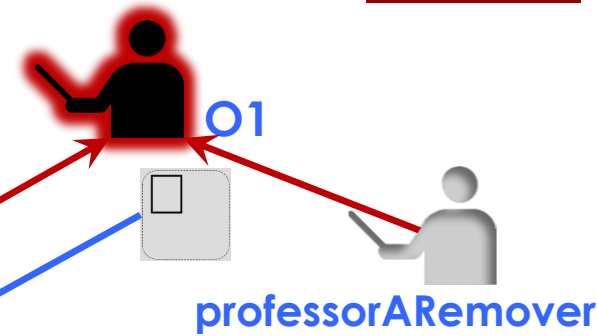
C Aula

```
public void desassociarProfessor() {  
    if (professor == null) {  
        return;  
    }  
    Professor professorARemover = professor;  
    professor = null;  
    professorARemover.remover(this);  
}
```



C Professor

```
public void remover(Aula aula) {  
}
```



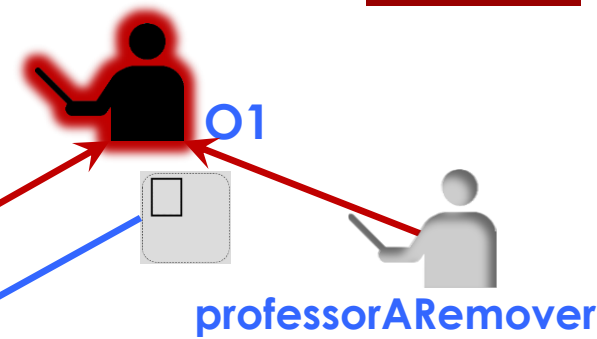
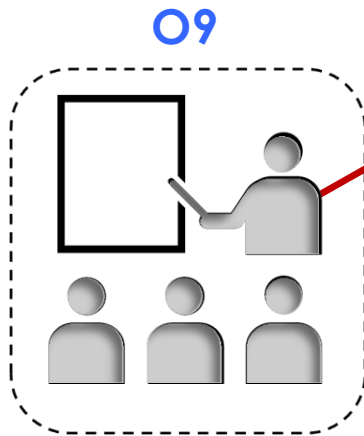
COMO SE REMOVE UMA
AULA DE UM PROFESSOR?

2.4. EXEMPLO

64

Aula

```
public void desassociarProfessor() {  
    if (professor == null) {  
        return;  
    }  
    Professor professorARemover = professor;  
    professor = null;  
    professorARemover.remover(this);  
}
```



Professor

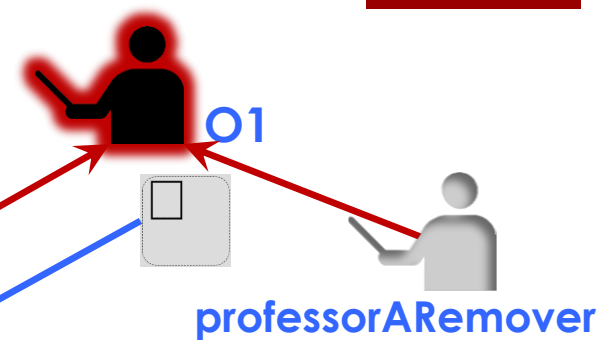
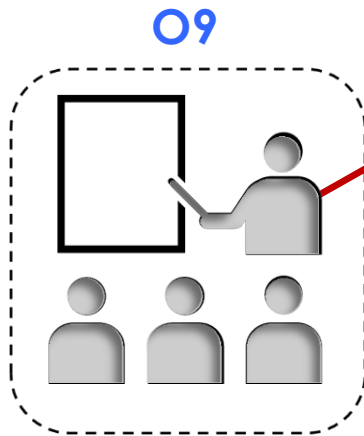
```
public void remover(Aula aula) {  
    if (!aulas.contains(aula)) {  
        return;  
    }  
    aulas.remove(aula);  
    aula.desassociarProfessor();  
}
```


2.4. EXEMPLO

65

Aula

```
public void desassociarProfessor() {  
    if (professor == null) {  
        return;  
    }  
    Professor professorARemover = professor;  
    professor = null;  
    professorARemover.remover(this);  
}
```



Professor

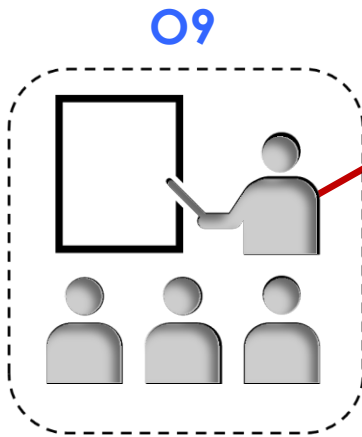
```
public void remover(Aula aula) {  
    if (!aulas.contains(aula)) {  
        return;  
    }  
    aulas.remove(aula);  
    aula.desassociarProfessor();  
}
```

2.4. EXEMPLO

66

Aula

```
public void desassociarProfessor() {  
    if (professor == null) {  
        return;  
    }  
    Professor professorARemover = professor;  
    professor = null;  
    professorARemover.remover(this);  
}
```



Professor

```
public void remover(Aula aula) {  
    if (!aulas.contains(aula)) {  
        return;  
    }  
    aulas.remove(aula);  
    aula.desassociarProfessor();  
}
```

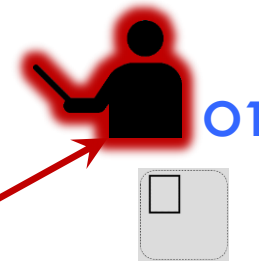
O PROFESSOR NÃO É DESASSOCIADO DE
UMA AULA SEM PROFESSOR ASSOCIADO

2.4. EXEMPLO

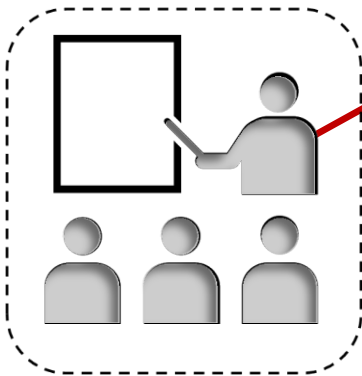
67

Aula

```
public void setProfessor(Professor professor) {  
    if (professor == null || this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



O9



Professor

```
public void adicionar(Aula aula) {  
}
```

COMO SE ADICIONA UMA
AULA A UM PROFESSOR?

2.4. EXEMPLO

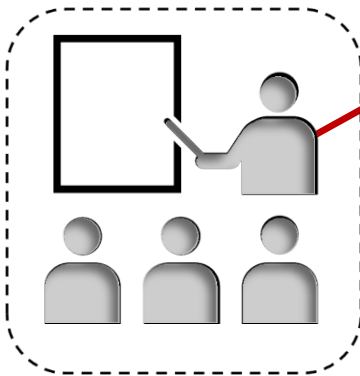
68

Aula

```
public void setProfessor(Professor professor) {  
    if (professor == null || this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



O9



Professor

```
public void adicionar(Aula aula) {  
    if (aula == null || aulas.contains(aula)) {  
        return;  
    }  
    aulas.add(aula);  
    aula.setProfessor(this);  
}
```

COMO SE ADICIONA UMA
AULA A UM PROFESSOR?

2.4. EXEMPLO

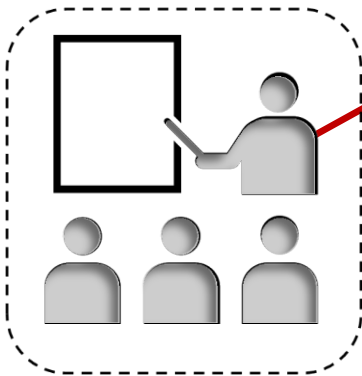
69

Aula

```
public void setProfessor(Professor professor) {  
    if (professor == null || this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



O9



Professor

```
public void adicionar(Aula aula) {  
    if (aula == null || aulas.contains(aula)) {  
        return;  
    }  
    aulas.add(aula);  
    aula.setProfessor(this);  
}
```

2.4. EXEMPLO

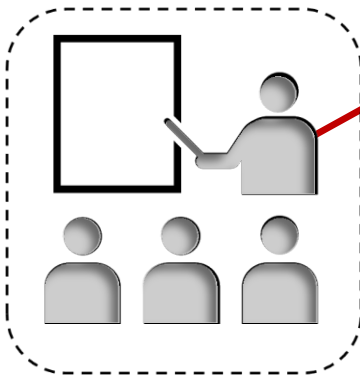
70

Aula

```
public void setProfessor(Professor professor) {  
    if (professor == null || this.professor == professor) {  
        return;  
    }  
    if (this.professor != null) {  
        this.professor.remover(this);  
    }  
    this.professor = professor;  
    this.professor.adicionar(this);  
}
```



O9



Professor

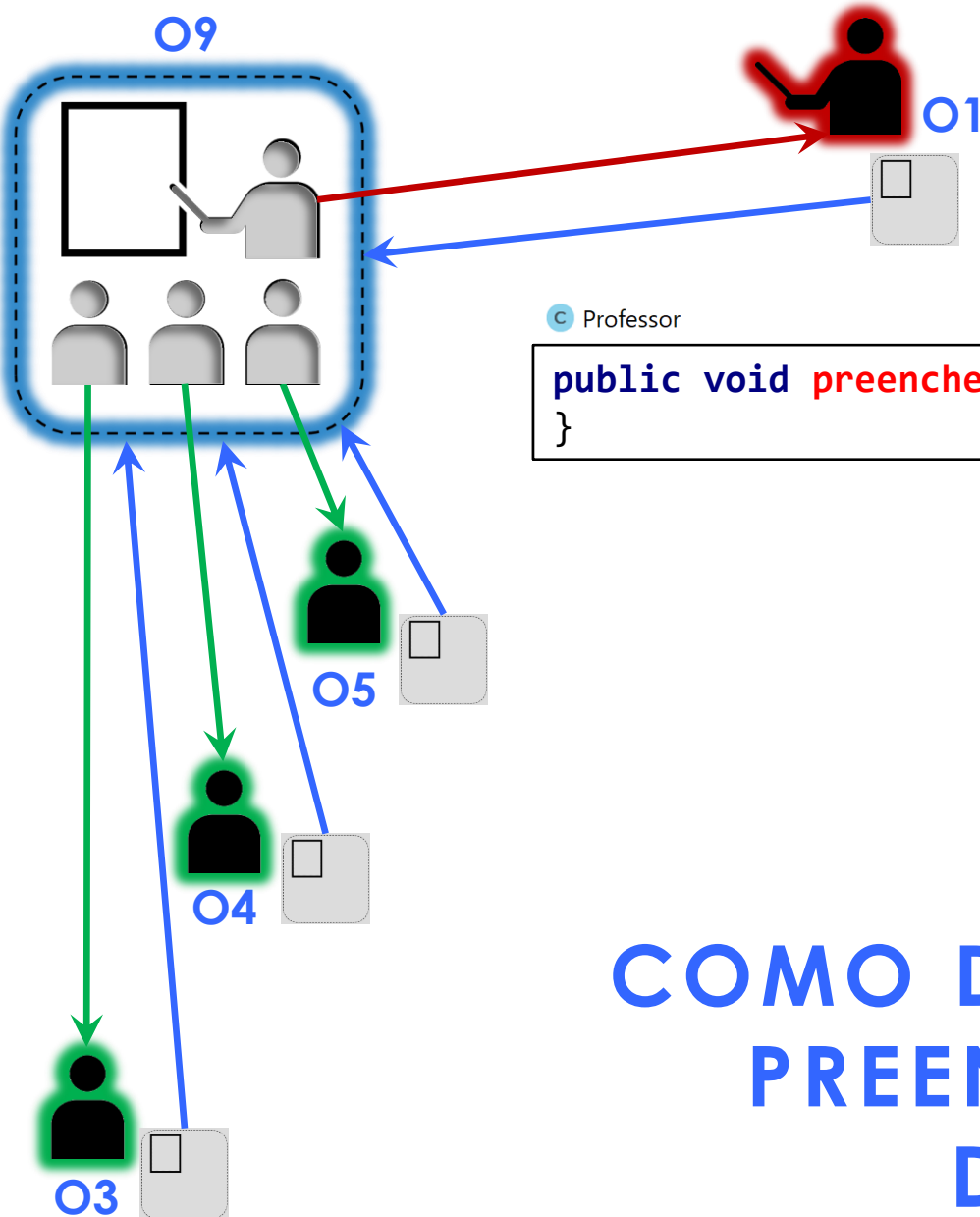
```
public void adicionar(Aula aula) {  
    if (aula == null || aulas.contains(aula)) {  
        return;  
    }  
    aulas.add(aula);  
    aula.setProfessor(this);  
}
```

O PROFESSOR NÃO É ASSOCIADO MAIS
DO QUE UMA VEZ À MESMA AULA

**Vamos agora
preencher o sumário
de uma aula**

2.4. EXEMPLO

72



Main

```
public Main() {  
    ...  
    o1.preencherSumario(o9);  
}
```

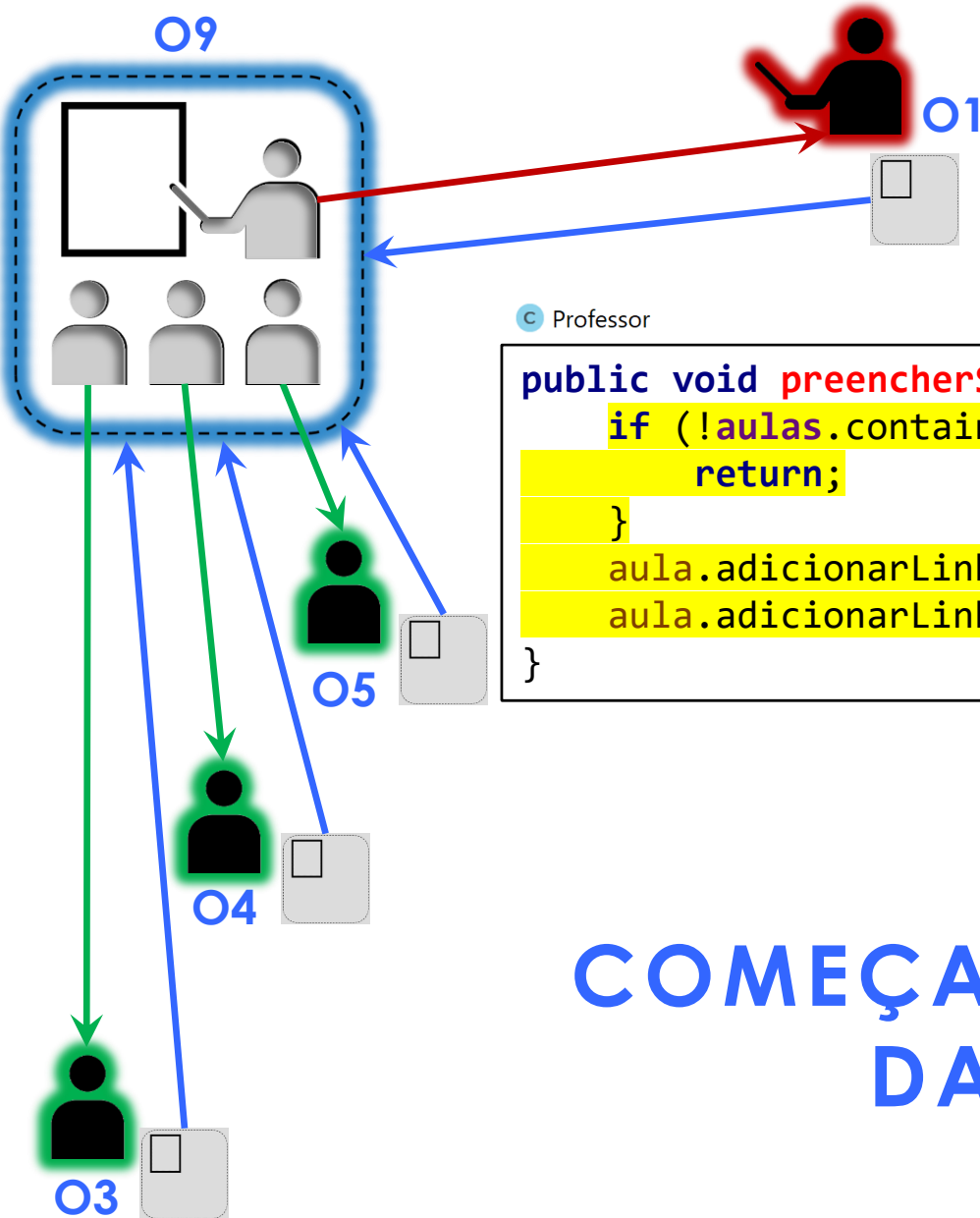
Professor

```
public void preencherSumario(Aula aula) {  
}
```

COMO DEVE, UM PROFESSOR,
PREENCHER O SUMÁRIO
DE UMA AULA?

2.4. EXEMPLO

73



Main

```
public Main() {  
    ...  
    o1.preencherSumario(o9);  
}
```

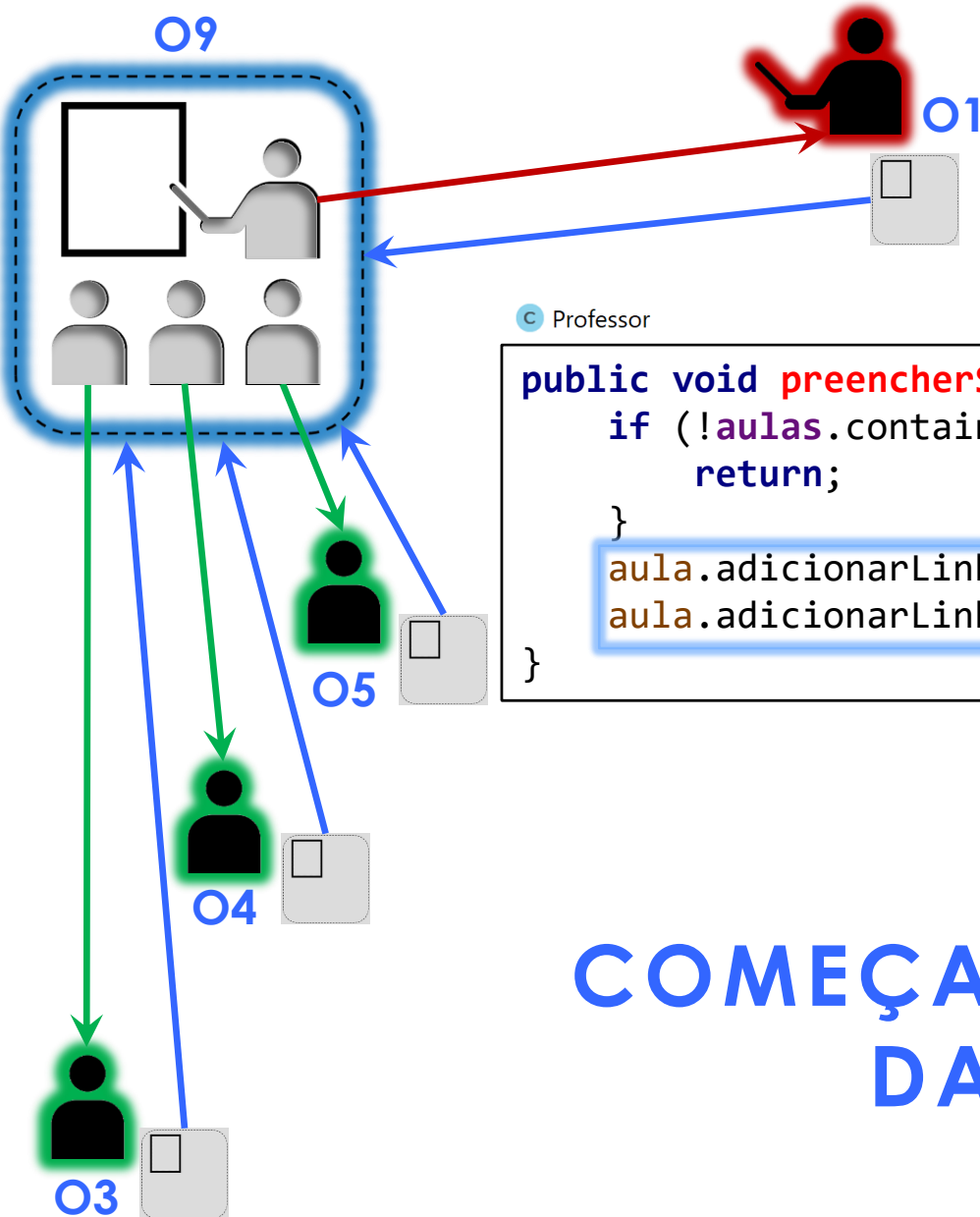
Professor

```
public void preencherSumario(Aula aula) {  
    if (!aulas.contains(aula)) {  
        return;  
    }  
    aula.adicionarLinhaSumario(aula.getNome());  
    aula.adicionarLinhaSumario(String.valueOf(aula.getNumero()));  
}
```

COMEÇA POR PREENCHER OS
DADOS DA AULA

2.4. EXEMPLO

74



C Aula

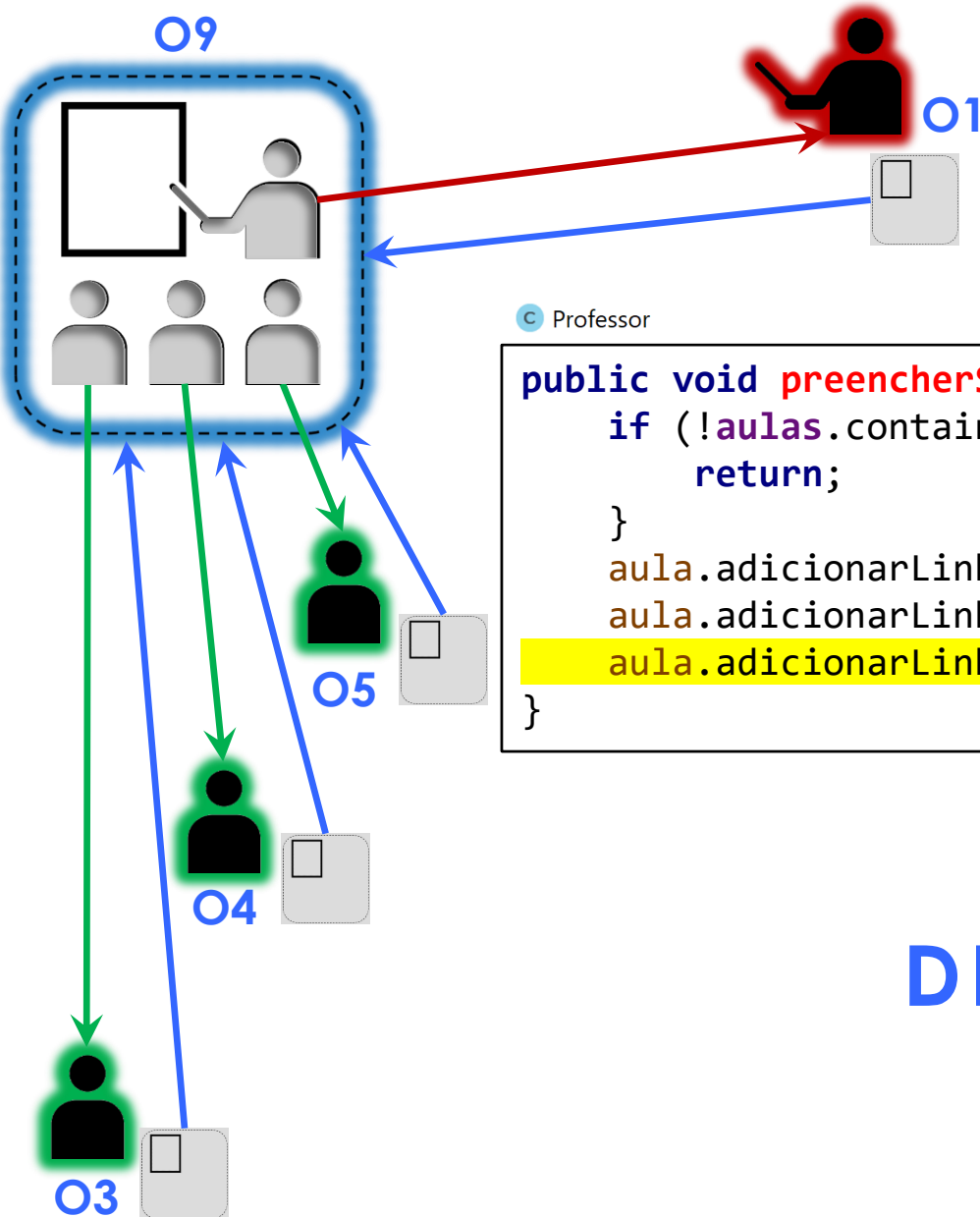
```
public void adicionarLinhaSumario(String linha) {  
    StringBuilder sb = new StringBuilder(sumario);  
    sb.append(linha).append("\n");  
    sumario = sb.toString();  
}
```

C Professor

```
public void preencherSumario(Aula aula) {  
    if (!aulas.contains(aula)) {  
        return;  
    }  
    aula.adicionarLinhaSumario(aula.getNome());  
    aula.adicionarLinhaSumario(String.valueOf(aula.getNumero()));  
}
```

COMEÇA POR PREENCHER OS
DADOS DA AULA

2.4. EXEMPLE

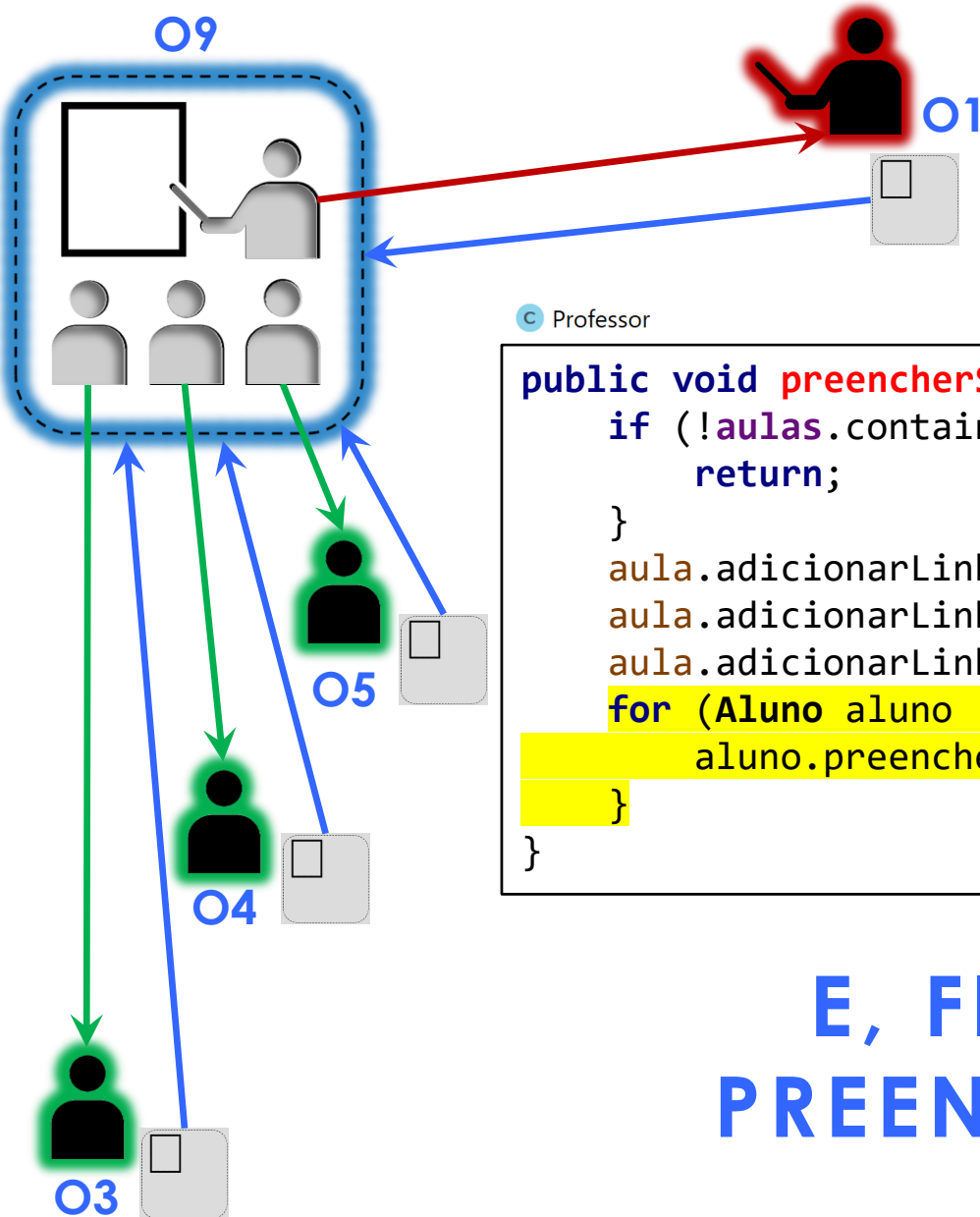


C Professor

```
public void preencherSumario(Aula aula) {
    if (!aulas.contains(aula)) {
        return;
    }
    aula.adicionarLinhaSumario(aula.getNome());
    aula.adicionarLinhaSumario(String.valueOf(aula.getNumero()));
    aula.adicionarLinhaSumario(nome);
}
```

DEPOIS ASSINA

2.4. EXEMPLE



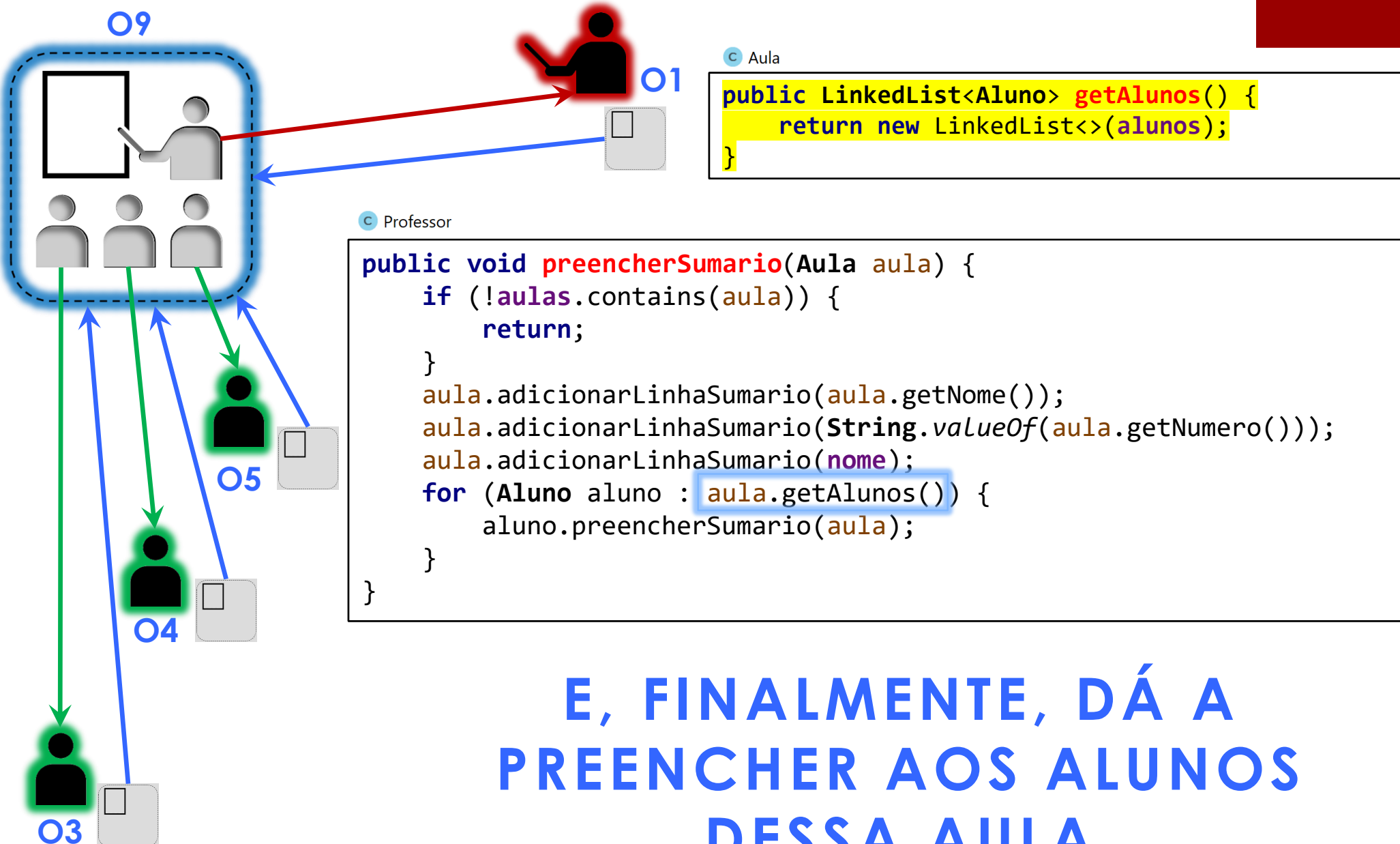
C Professor

```
public void preencherSumario(Aula aula) {
    if (!aulas.contains(aula)) {
        return;
    }
    aula.adicionarLinhaSumario(aula.getNome());
    aula.adicionarLinhaSumario(String.valueOf(aula.getNumero()));
    aula.adicionarLinhaSumario(nome);
    for (Aluno aluno : aula.getAlunos()) {
        aluno.preencherSumario(aula);
    }
}
```

**E, FINALMENTE, DÁ A
PREENCHER AOS ALUNOS
DESSA AULA**

2.4. EXEMPLO

77



E, FINALMENTE, DÁ A
PREENCHER AOS ALUNOS
DESSA AULA

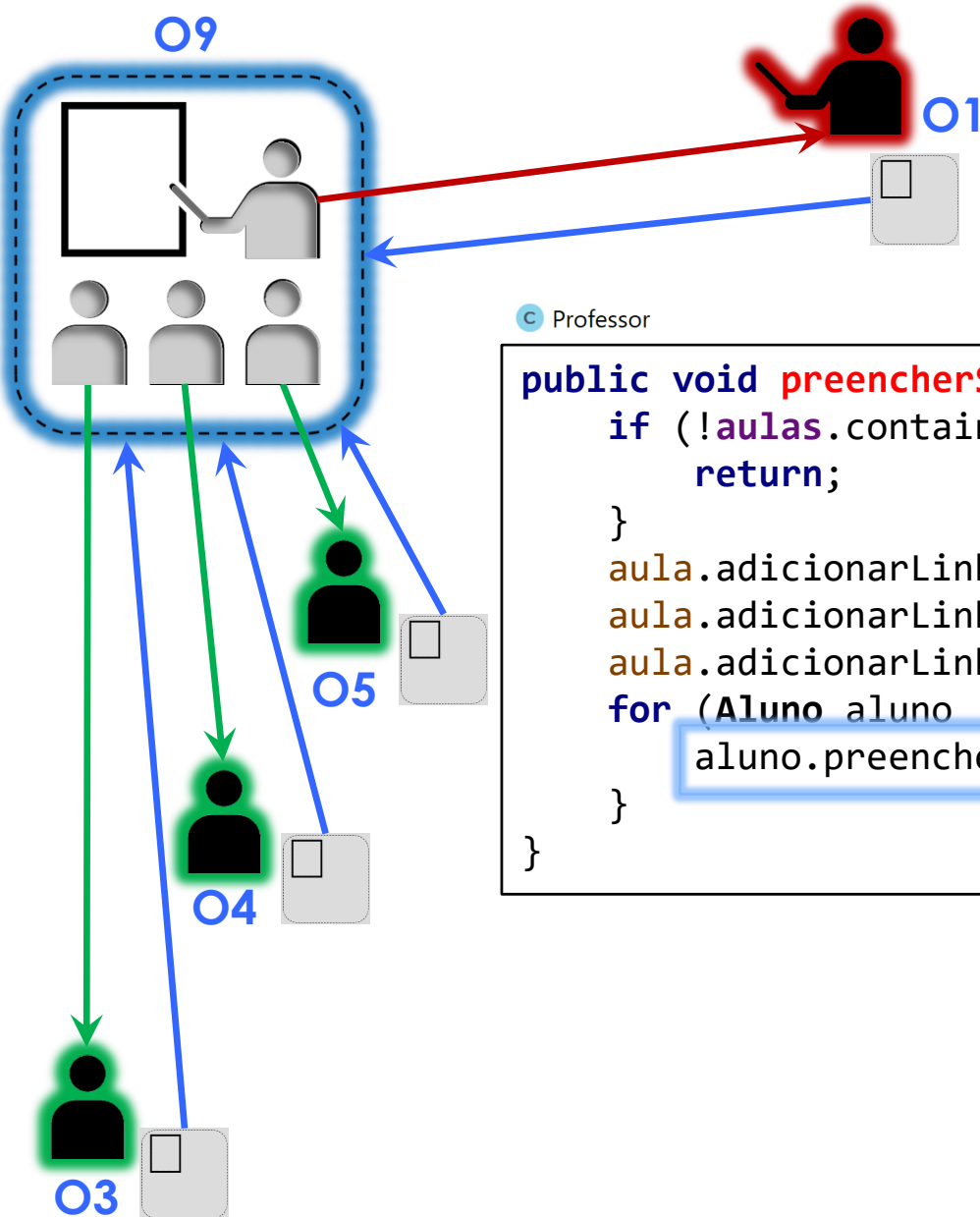
2.4. EXEMPLO

78



DEVOLUÇÃO DE UMA NOVA
LISTA COM AS REFERÊNCIAS
PARA OS ALUNOS DA AULA

2.4. EXEMPLE



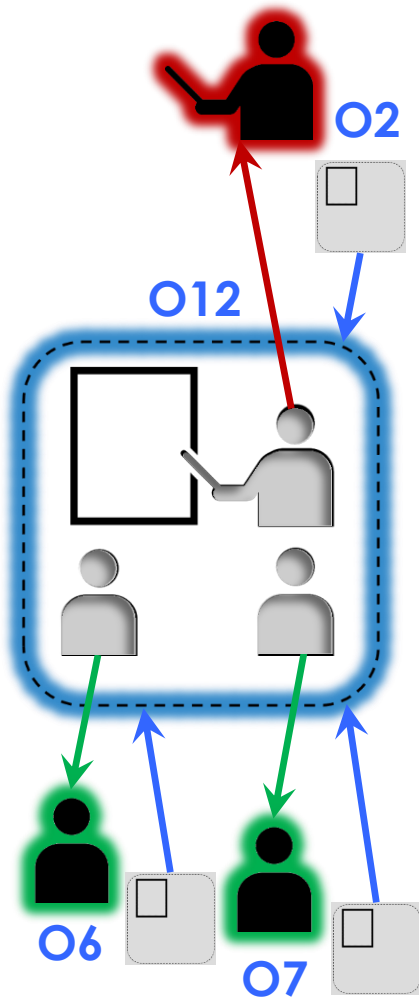
C Aluno

```
public void preencherSumario(Aula aula) {
    if (!aulas.contains(aula)) {
        return;
    }
    aula.adicionarLinhaSumario(nome);
}
```

C Professor

```
public void preencherSumario(Aula aula) {
    if (!aulas.contains(aula)) {
        return;
    }
    aula.adicionarLinhaSumario(aula.getNome());
    aula.adicionarLinhaSumario(String.valueOf(aula.getNumero()));
    aula.adicionarLinhaSumario(nome);
    for (Aluno aluno : aula.getAlunos()) {
        aluno.preencherSumario(aula);
    }
}
```

**E se quiséssemos criar
aulas já com professor
e/ou alunos?**

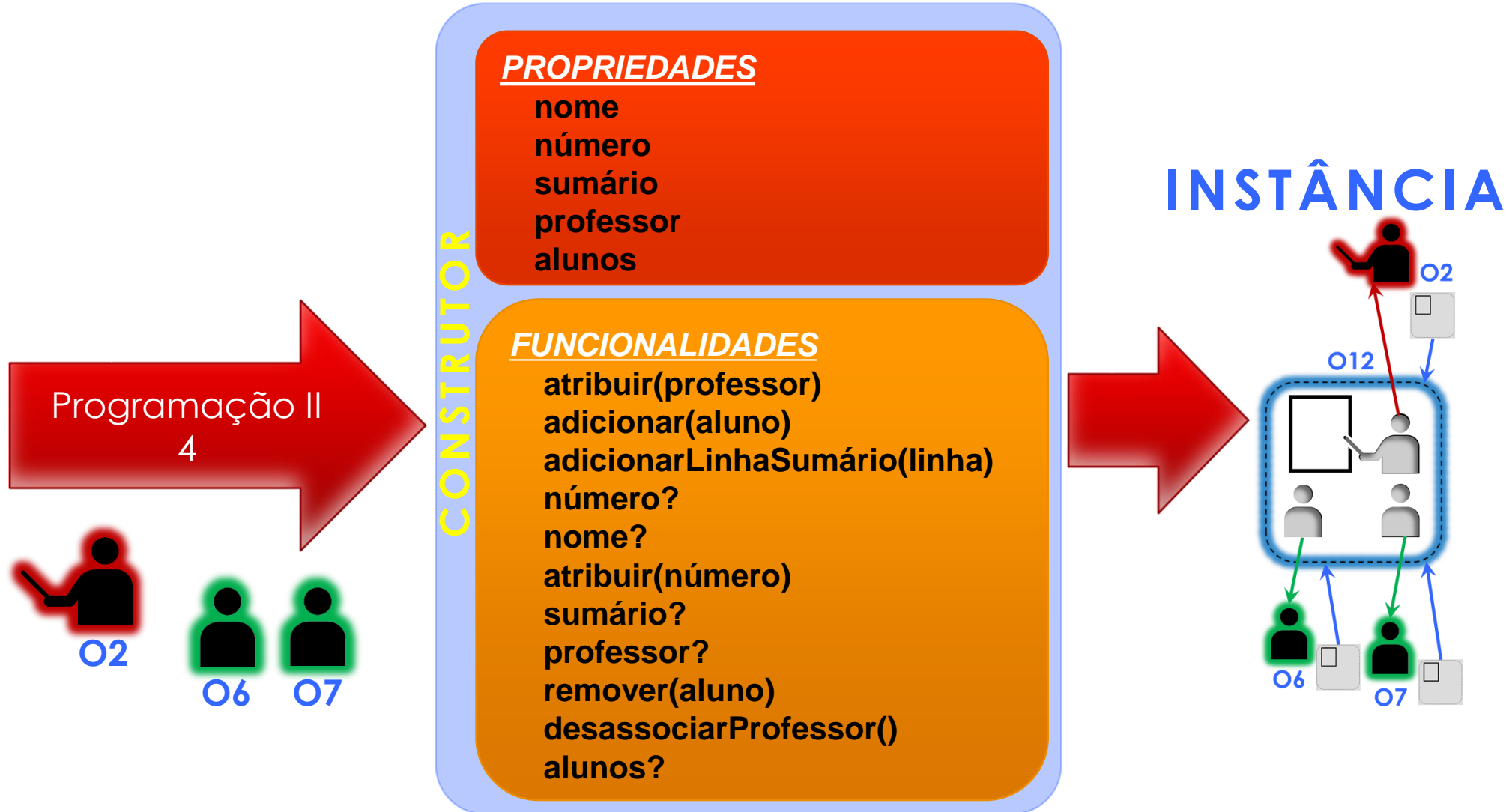


2.5. OVERLOADING DE MÉTODOS

INSTANCIACÃO

82

aula



2.5. OVERLOADING DE MÉTODOS

83

```
public class Aula {  
    private String nome;  
    private long numero;  
    private String sumario;  
    private Professor professor;  
    private LinkedList<Aluno> alunos;
```

OVERLOADING DE MÉTODOS IGUAL IDENTIFICADOR MAS ASSINATURAS DIFERENTES

```
    public Aula(String nome, long numero) {  
        this.nome = nome;  
        this.numero = numero;  
        this.sumario = "";  
        this.professor = null;  
        this.alunos = new LinkedList<>();  
    }
```

```
    public Aula(String nome, long numero, Professor professor, LinkedList<Aluno> alunos) {  
        this.nome = nome;  
        this.numero = numero;  
        this.sumario = "";  
        setProfessor(professor);  
        this.alunos = new LinkedList<>();  
        for (Aluno aluno : alunos) {  
            adicionar(aluno);  
        }  
    }
```

...

```
}
```

2.5. OVERLOADING DE MÉTODOS

84

```
public class Aula {  
    private String nome;  
    private long numero;  
    private String sumario;  
    private Professor professor;  
    private LinkedList<Aluno> alunos;
```

O CONSTRUTOR ESPECÍFICO
DEVE INVOCAR O
CONSTRUTOR MAIS GERAL

```
public Aula(String nome, long numero) {  
    this.nome = nome;  
    this.numero = numero;  
    this.sumario = "";  
    this.professor = null;  
    this.alunos = new LinkedList<>();  
}
```

```
public Aula(String nome, long numero, Professor professor, LinkedList<Aluno> alunos) {  
    this.nome = nome;  
    this.numero = numero;  
    this.sumario = "";  
    setProfessor(professor);  
    this.alunos = new LinkedList<>();  
    for (Aluno aluno : alunos) {  
        adicionar(aluno);  
    }  
}  
...  
}
```

melhoramento

```
public Aula(String nome, long numero) {  
    this(nome, numero, null, new LinkedList<>());  
}
```

2.5. OVERLOADING DE MÉTODOS

85

```
public class Aula {  
    private String nome;  
    private long numero;  
    private String sumario;  
    private Professor professor;  
    private LinkedList<Aluno> alunos;
```

O CONSTRUTOR ESPECÍFICO
DEVE INVOCAR O
CONSTRUTOR MAIS GERAL

```
public Aula(String nome, long numero) {  
    this.nome = nome;  
    this.numero = numero;  
    this.sumario = "";  
    this.professor = null;  
    this.alunos = new LinkedList<>();  
}
```

```
public Aula(String nome, long numero, Professor professor, LinkedList<Aluno> alunos) {  
    this.nome = nome;  
    this.numero = numero;  
    this.sumario = "";  
    setProfessor(professor);  
    this.alunos = new LinkedList<>();  
    for (Aluno aluno : alunos) {  
        adicionar(aluno);  
    }  
}  
...  
}
```

melhoramento

invoca

```
public Aula(String nome, long numero) {  
    this(nome, numero, null, new LinkedList<>());  
}
```


2.5. OVERLOADING DE MÉTODOS

86

```
public class Aula {  
    private String nome;  
    private long numero;  
    private String sumario;  
    private Professor professor;  
    private LinkedList<Aluno> alunos;
```

```
    public Aula(String nome, long numero) {  
        this(nome, numero, null, new LinkedList<>());  
    }
```

```
    public Aula(String nome, long numero, Professor professor, LinkedList<Aluno> alunos) {  
        this.nome = nome;  
        this.numero = numero;  
        this.sumario = "";  
        setProfessor(professor);  
        this.alunos = new LinkedList<>();  
        for (Aluno aluno : alunos) {  
            adicionar(aluno);  
        }  
    }  
    ...  
}
```



2.5. OVERLOADING DE MÉTODOS

87

	classes	Professor	Aluno	Aula
tipo	atributo			
String	nome	x	x	x
long	numero	x	x	x
LinkedList<Aula>	aulas	x	x	
String	sumario			x
Professor	professor			x
LinkedList<Aluno>	alunos			x
return	método			
void	setProfessor(Professor)			x
void	adicionar(Aula)	x	x	
void	preencherSumario(Aula)	x	x	
void	adicionar(Aluno)			x
void	adicionarLinhaSumario(String)			x
String	getNome()	x	x	x
long	getNumero()	x	x	x
void	setNumero(long)	x	x	x
String	getSumario()			x
Professor	getProfessor()			x
void	desassociarProfessor()			x
void	remover(Aula)	x	x	
LinkedList<Aluno>	getAlunos()			x
void	remover(Aluno)			x

TABELA COM
CARACTERÍSTICAS
DAS CLASSES

2.5. OVERLOADING DE MÉTODOS

88

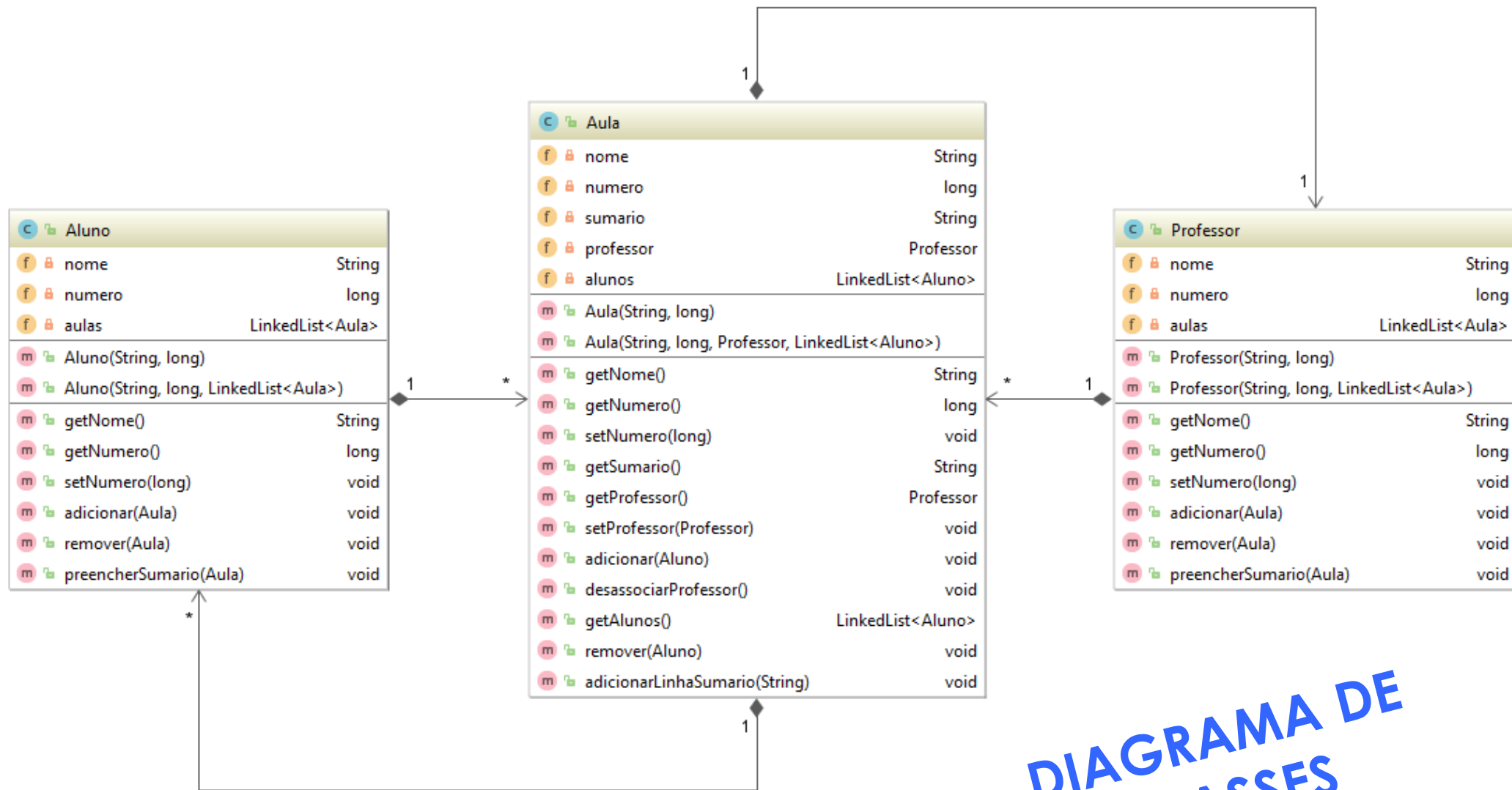


DIAGRAMA DE
CLASSES