

**INSTITUTO FEDERAL**  
Goiás

Instituto Federal de Goiás  
Câmpus Goiânia

Bacharelado em Sistemas de Informação  
Disciplina: Programação Orientada a Objetos I

# Associação de Classes

## Dependência

Prof. Ms. Dory Gonzaga Rodrigues  
Goiânia - GO

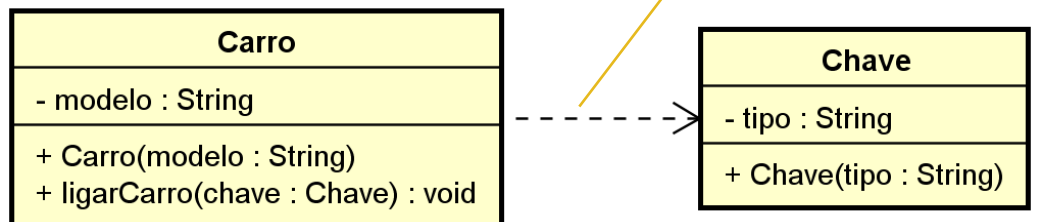
# Dependência

Uma dependência é um **relacionamento não estrutural** (de “uso”) que tenta apontar que mudanças em uma classes pode afetar outra classe que a usa, mas não necessariamente o inverso.



É caracterizada por **um objeto depender de outro** de forma temporária, onde provavelmente será :

- ▶ Um parâmetros de entrada de um método;
- ▶ Um tipo de retorno de um método;
- ▶ Utilizado dentro do código de métodos;
- ▶ Exceções lançadas.



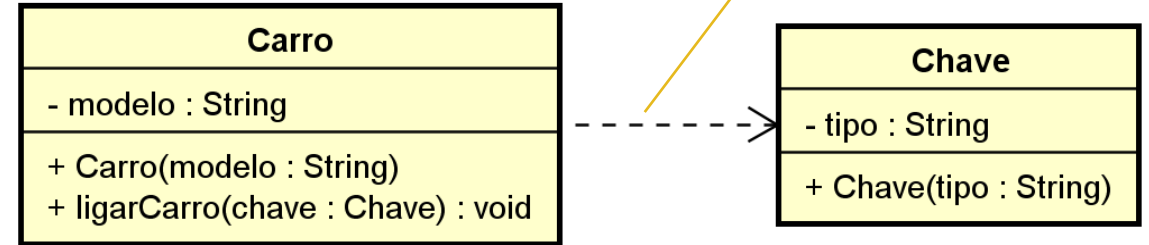
Um Carro “Depende” da Chave “Temporariamente”.



## Dependência

### Implementando a Classe Chave

```
public class Chave {  
  
    private String tipo;  
  
    public Chave(String tipo) {  
        this.tipo = tipo;  
    }  
  
    public String getTipo() {  
        return tipo;  
    }  
  
    @Override  
    public String toString() {  
        return "Chave [tipo=" + tipo + " ]";  
    }  
}
```



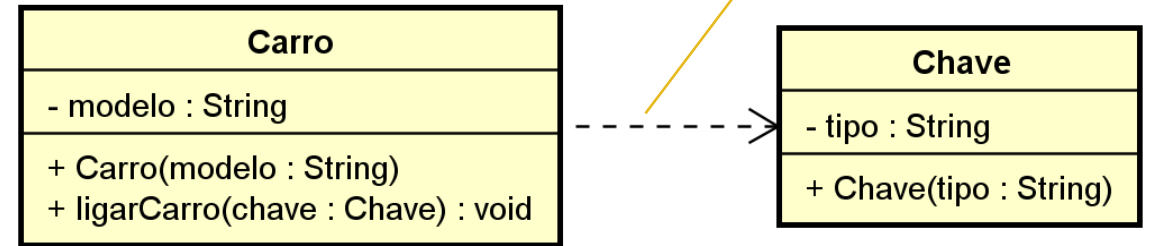
O sentido da seta indica que Carro depende de Chave.

Um Carro “Depende” da Chave “Temporariamente”.

## Dependência

### Implementando a Classe Carro

```
public class Carro {  
  
    private String modelo;  
    private boolean ligado;  
    private boolean andando;  
  
    public Carro(String modelo) {  
        this.modelo = modelo;  
    }  
  
    public void ligarCarro(Chave chave) {  
  
        if (chave == null) {  
            throw new NullPointerException("A Chave não pode ser nula!");  
        }  
        System.out.println("Usando a " + chave + " para ligar o carro.");  
        this.ligado = true;  
    }  
}
```



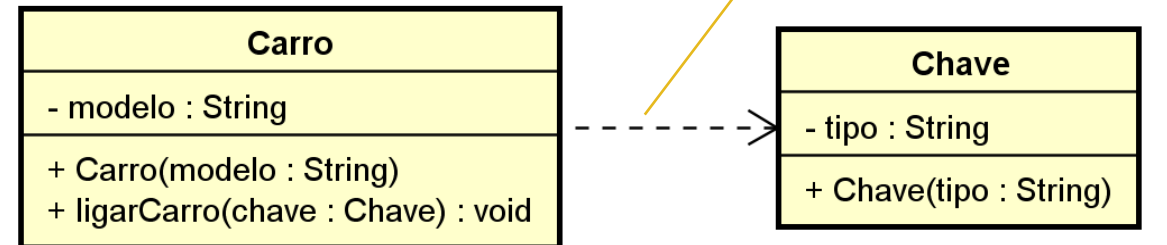
O sentido da seta indica que Carro depende de Chave.

Um Carro “Depende” da Chave “Temporariamente”.

# Dependência

## Implementando a Classe Carro

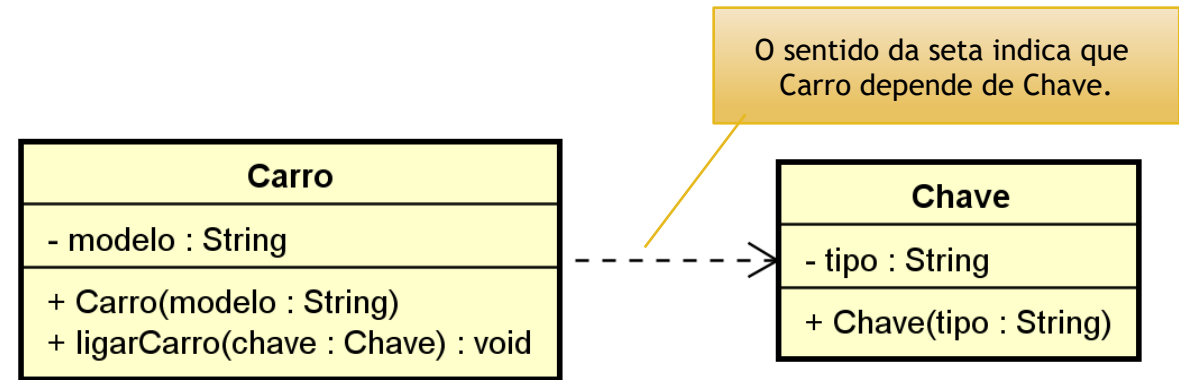
```
public boolean andar() {  
    if (ligado) {  
        System.out.println("O Carro está andando.");  
        andando = true;  
    } else  
        System.out.println("Não foi possível andar, pois o Carro está desligado.");  
    return andando;  
}  
  
public String getModelo() {  
    return modelo;  
}  
  
public boolean isLigado() {  
    return ligado;  
}  
  
public boolean isAndando() {  
    return andando;  
}  
  
@Override  
public String toString() {  
    return "Carro [modelo=" + modelo  
        + ", ligado=" + ligado  
        + ", andando=" + andando + "];"  
}  
}
```



Um Carro “Depende” da Chave “Temporariamente”.

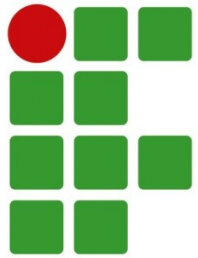
## Dependência Programa Principal

```
public class TesteCarro {  
  
    public static void main(String[] args) {  
  
        Chave chave = new Chave("ChaveKia");  
        Carro carro = new Carro("Sorento");  
  
        System.out.println(carro);  
  
        carro.andar();  
  
        carro.ligarCarro(chave);  
        carro.andar();  
  
        System.out.println(carro);  
    }  
}
```



Um Carro "Depende" da Chave "Temporariamente".

```
Saída - Aula15 (run) ×  
  
run:  
Carro [modelo=Sorento, ligado=false, andando=false]  
Não foi possível andar, pois o Carro está desligado.  
Usando a Chave [tipo=ChaveKia] para ligar o carro.  
O Carro está andando.  
Carro [modelo=Sorento, ligado=true, andando=true]  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



**INSTITUTO FEDERAL**  
Goiás

Instituto Federal de Goiás  
Câmpus Goiânia

Bacharelado em Sistemas de Informação  
Disciplina: Programação Orientada a Objetos I

# Associação de Classes

## Classe de Associação

Prof. Ms. Dory Gonzaga Rodrigues  
Goiânia - GO

# Classe de Associação

É uma classe que está ligada a uma associação, em vez de estar ligada a outras classes.

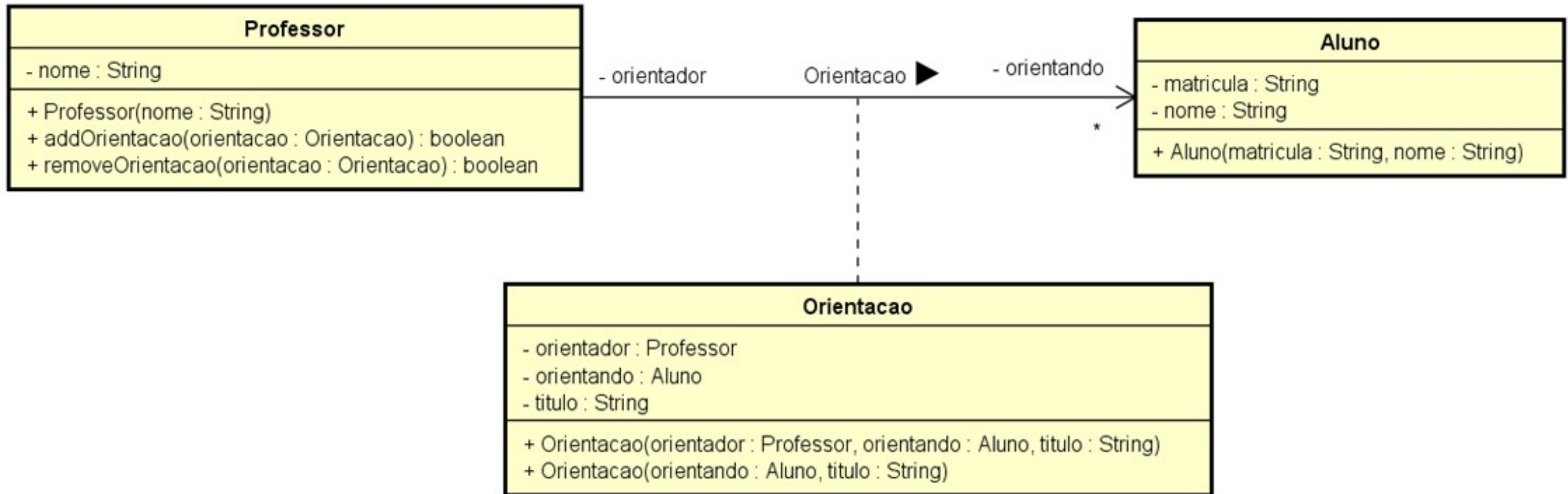


É normalmente necessária quando duas ou mais classes estão associadas, existindo atributos que não encaixam em nenhuma delas, sendo necessário manter informações sobre esta associação.



# Classe de Associação

É uma classe que está ligada a uma associação, em vez de estar ligada a outras classes.



# Classe de Associação

## Implementando a Classe Aluno

Aluno
- matricula : String - nome : String
+ Aluno(matricula : String, nome : String)

```
1 public class Aluno {
2
3     private String matricula;
4     private String nome;
5
6     public Aluno(String matricula, String nome) {
7         this.matricula = matricula;
8         this.nome = nome;
9     }
10
11     public String getMatricula() {
12         return matricula;
13     }
14
15     public void setMatricula(String matricula) {
16         this.matricula = matricula;
17     }
18
19     public String getNome() {
20         return nome;
21     }
22
23     public void setNome(String nome) {
24         this.nome = nome;
25     }
26 }
```

```
27 @Override
28 public boolean equals(Object obj) {
29     if (this == obj)
30         return true;
31     if (obj == null)
32         return false;
33     if (getClass() != obj.getClass())
34         return false;
35     Aluno other = (Aluno) obj;
36     if (matricula == null) {
37         if (other.matricula != null)
38             return false;
39     } else if (!matricula.equals(other.matricula))
40         return false;
41     if (nome == null) {
42         if (other.nome != null)
43             return false;
44     } else if (!nome.equals(other.nome))
45         return false;
46     return true;
47 }
```

CONTINUA >>

# Classe de Associação

## Implementando a Classe Aluno

Aluno
- matricula : String - nome : String
+ Aluno(matricula : String, nome : String)

```
49 @Override
50 public int hashCode() {
51     final int prime = 31;
52     int result = 1;
53     result = prime * result + ((matricula == null) ? 0 : matricula.hashCode());
54     result = prime * result + ((nome == null) ? 0 : nome.hashCode());
55     return result;
56 }
57
58 @Override
59 public String toString() {
60     return "Aluno [matricula=" + matricula + ", nome=" + nome + "]";
61 }
62 }
```

# Classe de Associação

## Implementando a Classe Professor

```
1 import java.util.ArrayList;
2
3
4 public class Professor {
5
6     private String nome;
7     private List<Orientacao> listaOrientacao = new ArrayList<>();
8
9     public Professor(String nome) {
10         super();
11         this.nome = nome;
12     }
13
14     public String getNome() {
15         return nome;
16     }
17
18     public void setNome(String nome) {
19         this.nome = nome;
20     }
21
22     public List<Orientacao> getListaOrientacao() {
23         return listaOrientacao;
24     }
25 }
```

Professor
- nome : String
+ Professor(nome : String) + addOrientacao(orientacao : Orientacao) : boolean + removeOrientacao(orientacao : Orientacao) : boolean

# Classe de Associação

## Implementando a Classe Professor

```
26 public boolean addOrientacao(Orientacao o) {  
27     boolean sucesso = false;  
28     if (o != null && !listaOrientacao.contains(o)) {  
29         this.listaOrientacao.add(o);  
30         o.setOrientador(this);  
31         sucesso = true;  
32     }  
33     return sucesso;  
34 }  
35  
36 public boolean removeOrientacao(Orientacao o) {  
37     boolean sucesso = false;  
38     if ( o != null && listaOrientacao.size() > 0 && listaOrientacao.contains(o) ) {  
39         listaOrientacao.remove(o);  
40         o.setOrientador(null);  
41         sucesso = true;  
42     }  
43     return sucesso;  
44 }
```

Professor
- nome : String
+ Professor(nome : String) + addOrientacao(orientacao : Orientacao) : boolean + removeOrientacao(orientacao : Orientacao) : boolean

# Classe de Associação


## Implementando a Classe Professor

Professor
- nome : String
+ Professor(nome : String) + addOrientacao(orientacao : Orientacao) : boolean + removeOrientacao(orientacao : Orientacao) : boolean

```
46⊖ @Override
47 public int hashCode() {
48     final int prime = 31;
49     int result = 1;
50     result = prime * result + ((nome == null) ? 0 : nome.hashCode());
51     return result;
52 }
```

```
54⊖ @Override
55 public boolean equals(Object obj) {
56     if (this == obj)
57         return true;
58     if (obj == null)
59         return false;
60     if (getClass() != obj.getClass())
61         return false;
62     Professor other = (Professor) obj;
63     if (nome == null) {
64         if (other.nome != null)
65             return false;
66     } else if (!nome.equals(other.nome))
67         return false;
68     return true;
69 }
```

```
71⊖ @Override
72 public String toString() {
73     return "Professor [nome=" + nome + "]";
74 }
75 }
```



# Classe de Associação

## Implementando a Classe Orientação

```
1
2 public class Orientacao {
3
4     private Professor orientador = null;
5     private Aluno orientando = null;
6     private String titulo = null;
7
8     public Orientacao(Professor orientador, Aluno orientando, String titulo) {
9
10         this.orientador = orientador;
11         this.orientador.addOrientacao(this);
12
13         this.orientando = orientando;
14
15         this.titulo = titulo;
16     }
17
18     public Orientacao(Aluno orientando, String titulo) {
19
20         this.orientando = orientando;
21         this.titulo = titulo;
22     }
23
24     public Professor getOrientador() {
25         return orientador;
26     }
27 }
```

Orientacao
- orientador : Professor - orientando : Aluno - titulo : String
+ Orientacao(orientador : Professor, orientando : Aluno, titulo : String) + Orientacao(orientando : Aluno, titulo : String)

# Classe de Associação

## Implementando a Classe Orientação

```
28 public void setOrientador(Professor orientador) {
29     this.orientador = orientador;
30 }
31
32 public Aluno getOrientando() {
33     return orientando;
34 }
35
36 public void setOrientando(Aluno orientando) {
37     this.orientando = orientando;
38 }
39
40 public String getTitulo() {
41     return titulo;
42 }
43
44 public void setTitulo(String titulo) {
45     this.titulo = titulo;
46 }
47
48 @Override
49 public int hashCode() {
50     final int prime = 31;
51     int result = 1;
52     result = prime * result + ((orientador == null) ? 0 : orientador.hashCode());
53     result = prime * result + ((orientando == null) ? 0 : orientando.hashCode());
54     result = prime * result + ((titulo == null) ? 0 : titulo.hashCode());
55     return result;
56 }
```

Orientacao
- orientador : Professor - orientando : Aluno - titulo : String
+ Orientacao(orientador : Professor, orientando : Aluno, titulo : String) + Orientacao(orientando : Aluno, titulo : String)



# Classe de Associação

## Implementando a Classe Orientação

```
58 @Override
59 public boolean equals(Object obj) {
60     if (this == obj)
61         return true;
62     if (obj == null)
63         return false;
64     if (getClass() != obj.getClass())
65         return false;
66     Orientacao other = (Orientacao) obj;
67     if (orientador == null) {
68         if (other.orientador != null)
69             return false;
70     } else if (!orientador.equals(other.orientador))
71         return false;
72     if (orientando == null) {
73         if (other.orientando != null)
74             return false;
75     } else if (!orientando.equals(other.orientando))
76         return false;
77     if (titulo == null) {
78         if (other.titulo != null)
79             return false;
80     } else if (!titulo.equals(other.titulo))
81         return false;
82     return true;
83 }
```

Orientacao
- orientador : Professor - orientando : Aluno - titulo : String
+ Orientacao(orientador : Professor, orientando : Aluno, titulo : String) + Orientacao(orientando : Aluno, titulo : String)

# Classe de Associação

## Implementando a Classe Orientação

```
85 @Override
86 public String toString() {
87     return "Orientacao [orientador=" + this.orientador +
88         ", orientando=" + this.orientando + ", titulo=" + this.titulo + "];"
89 }
90 }
```

Orientacao
- orientador : Professor - orientando : Aluno - titulo : String
+ Orientacao(orientador : Professor, orientando : Aluno, titulo : String) + Orientacao(orientando : Aluno, titulo : String)

# Classe de Associação

## Implementando a Classe Principal

```
1 import java.util.List;
2
3 public class TestaOrientador {
4
5     public static void main(String[] args) {
6
7         Aluno a1 = new Aluno("1", "João");
8         Aluno a2 = new Aluno("2", "Maria");
9         System.out.println(a1);
10        System.out.println(a2);
11
12        System.out.println("-----");
13        Professor p1 = new Professor("Dory");
14        System.out.println(p1);
15
16        System.out.println("-----");
17        Orientacao o1 = new Orientacao(p1, a1, "PHP");
18        Orientacao o2 = new Orientacao(p1, a2, "POO");
19        System.out.println(o1.toString());
20        System.out.println(o2.toString());
21
22        System.out.println("-----");
23        System.out.println(p1);
24        List<Orientacao> lista = p1.getListaOrientacao();
25        for (int i = 0; i < lista.size(); i++) {
26            System.out.println(lista.get(i).getOrientando() + " , Tema [" + lista.get(i).getTitulo() + "]);
27        }
28    }
29 }
```

Console

```
<terminated> TestaOrientador [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (17 de out de 2019 02:27:29)
Aluno [matricula=1, nome=João]
Aluno [matricula=2, nome=Maria]
-----
Professor [nome=Dory]
-----
Orientacao [orientador=Professor [nome=Dory], orientando=Aluno [matricula=1, nome=João], titulo=PHP]
Orientacao [orientador=Professor [nome=Dory], orientando=Aluno [matricula=2, nome=Maria], titulo=POO]
-----
Professor [nome=Dory]
Aluno [matricula=1, nome=João] , Tema [PHP]
Aluno [matricula=2, nome=Maria] , Tema [POO]
```