



*Faculdade de
Tecnologia SENAI
“Roberto Mange”*

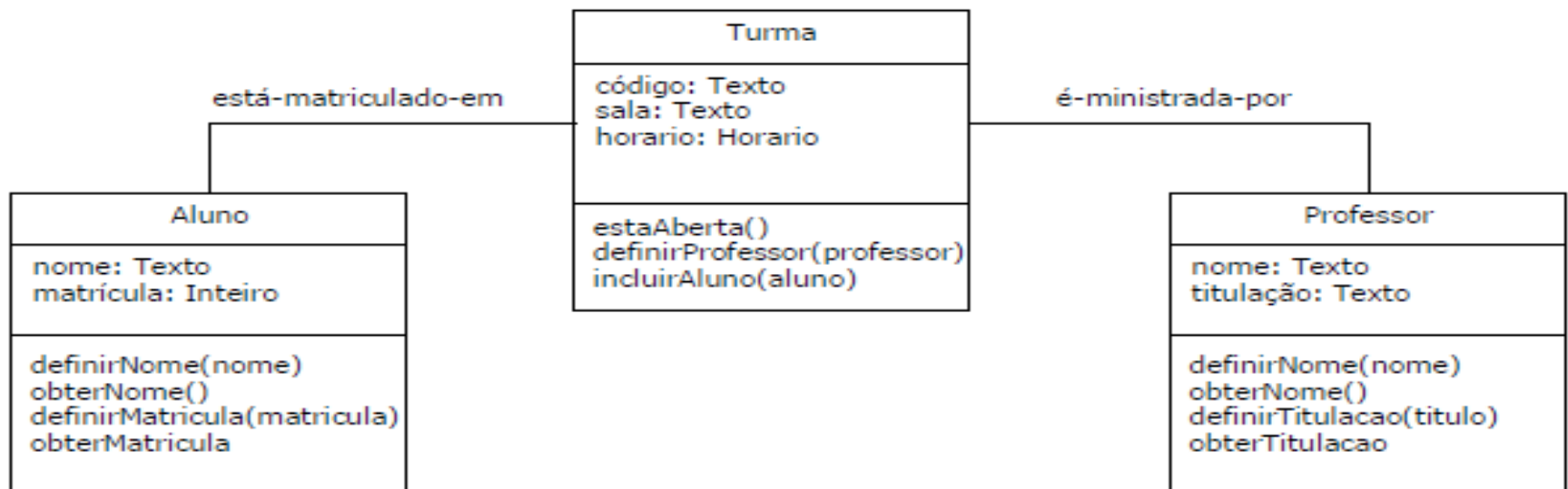


*Engenharia de Software
Modelagem - 2*

UML – Diagrama de classes

Introdução

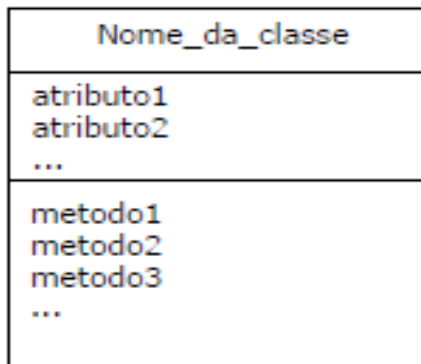
- Mostra um conjunto de classes e seus relacionamentos.
- É o diagrama central da modelagem orientada a objetos.



UML – Diagrama de classes

Classes

- Graficamente, as classes são representadas por retângulos incluindo nome, atributos e métodos.



- Devem receber nomes de acordo com o vocabulário do domínio do problema.
- É comum adotar um padrão para nomeá-las
Ex: todos os nomes de classes serão substantivos singulares com a primeira letra maiúscula

UML – Diagrama de classes

Classes

- Atributos
 - Representam o conjunto de características (estado) dos objetos daquela classe
 - Visibilidade:
 - + público: visível em qualquer classe de qualquer pacote
 - # protegido: visível para classes do mesmo pacote
 - privado: visível somente para classe

Exemplo:

+ nome : String



Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Classes

- Métodos
 - Representam o conjunto de operações (comportamento) que a classe fornece
 - Visibilidade:
 - + público: visível em qualquer classe de qualquer pacote
 - # protegido: visível para classes do mesmo pacote
 - privado: visível somente para classe

Exemplo:

- getNome() : String



Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Relacionamentos

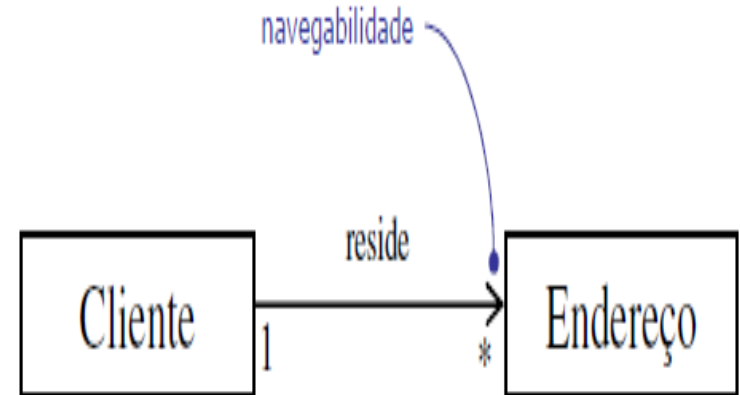
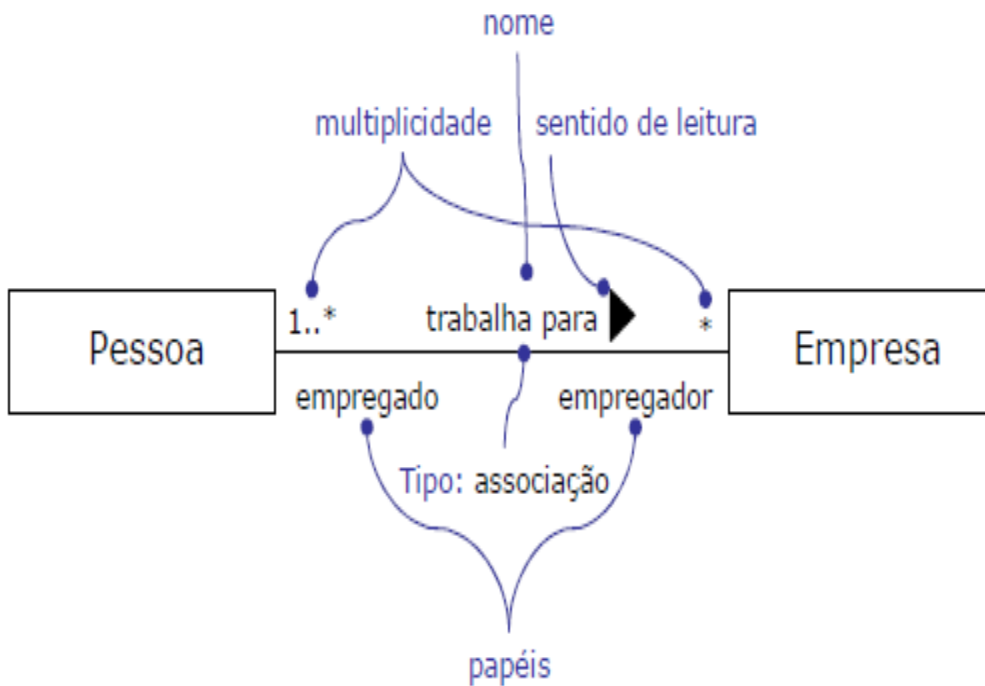
- Os relacionamentos possuem:
 - Nome: descrição dada ao relacionamento (faz, tem, possui,...)
 - Sentido de leitura
 - Navegabilidade: indicada por uma seta no fim do relacionamento
 - Multiplicidade: 0..1, 0..*, 1, 1..*, 2, 3..7
 - Tipo: associação (agregação, composição), generalização e dependência
 - Papéis: desempenhados por classes em um relacionamento



Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Relacionamentos



O cliente sabe quais são seus endereços, mas o endereço não sabe a quais clientes pertence

Graduação Tecnológica
Engenharia de Software



UML – Diagrama de classes Dependência

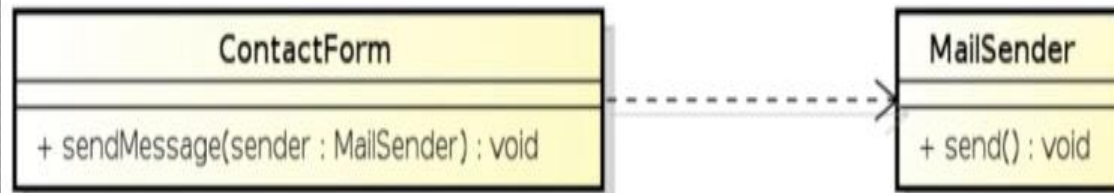
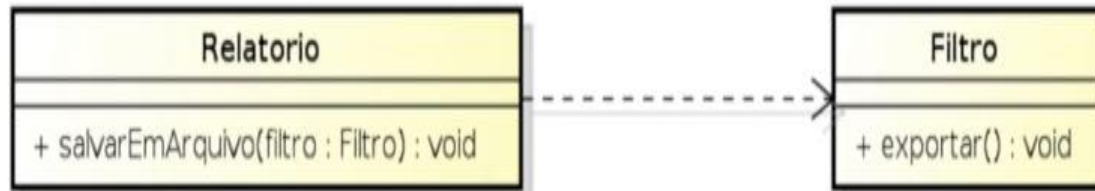
É o relacionamento mais simples entre classes/objetos. A **dependência** indica que um **objeto** **depende** da **especificação** de outro objeto.

Por especificação, podemos entender a **interface pública** do objeto (seu conjunto de métodos públicos).



*Graduação Tecnológica
Engenharia de Software*

UML – Diagrama de classes Dependência



As dependências devem ser usadas com critério, apesar de uma classe poder ter diversas dependências de outras, apenas se representa a mais importantes.

Graduação Tecnológica
Engenharia de Software

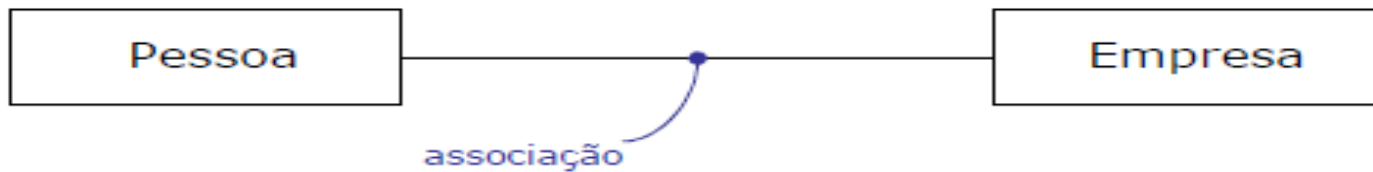


UML – Diagrama de classes

Associação

Uma **associação** é um relacionamento estrutural que indica que os objetos de uma classe estão vinculados a objetos de outra classe.

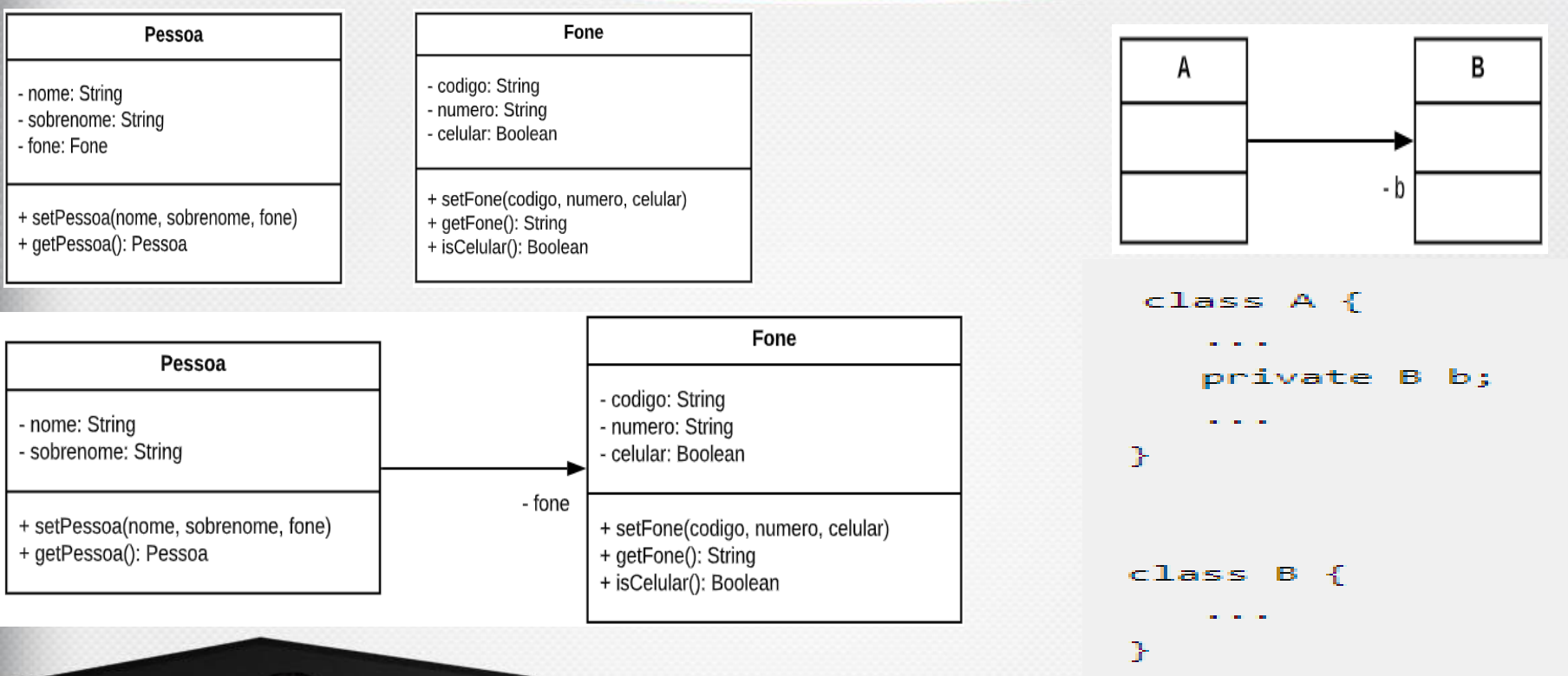
Uma associação é representada por uma linha sólida conectando duas classes.



Graduação Tecnológica
Engenharia de Software



UML – Diagrama de classes Associação



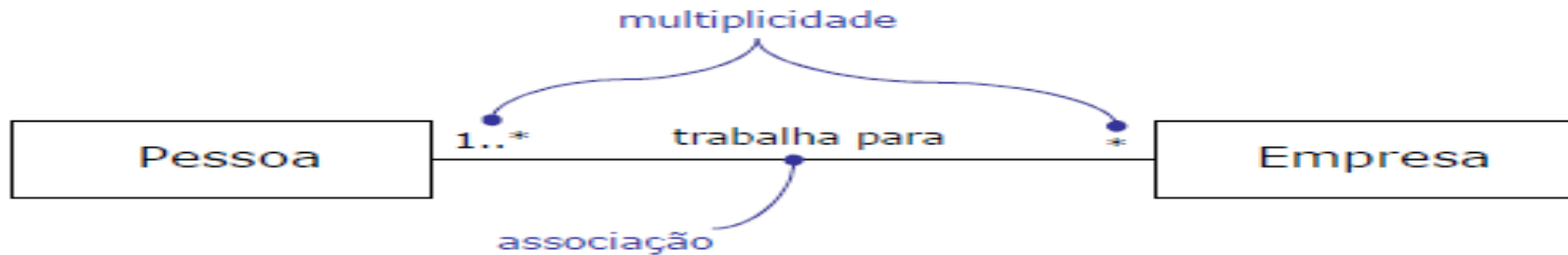
Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Associação

Indicadores de multiplicidade:

- 1 Exatamente um
- 1..* Um ou mais
- 0..* Zero ou mais (muitos)
- * Zero ou mais (muitos)
- 0..1 Zero ou um
- m..n Faixa de valores (por exemplo: 4..7)



Graduação Tecnológica
Engenharia de Software



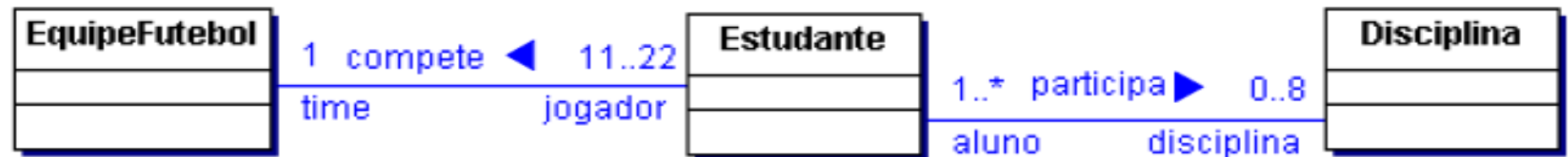
UML – Diagrama de classes

exemplo: Associação

Um **Estudante** pode ser
um **aluno** de uma Disciplina e
um **jogador** da Equipe de Futebol

Cada Disciplina deve ser cursada por no mínimo 1 aluno

Um aluno pode cursar de 0 até 8 disciplinas

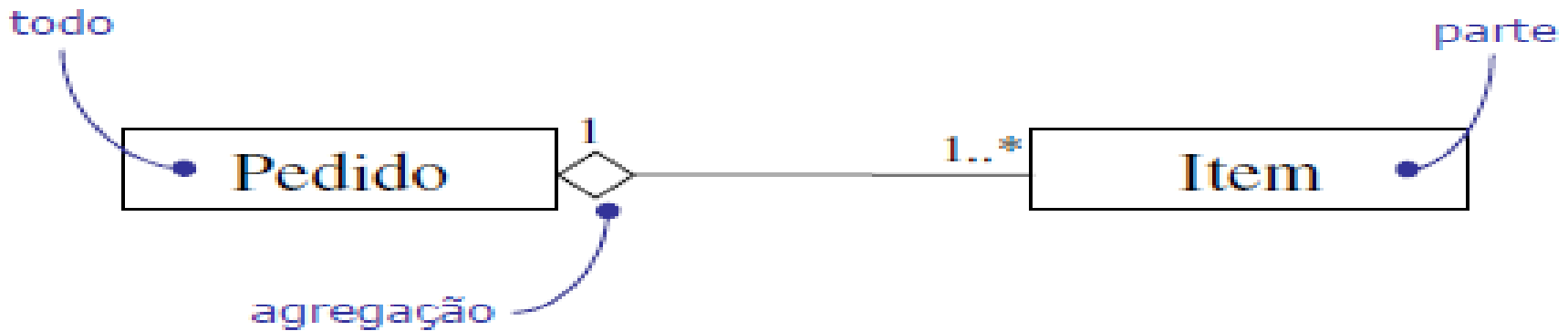


Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Associação de Agregação

É um tipo especial de associação
Utilizada para indicar “todo-parte”



um objeto “parte” pode fazer parte de vários objetos “todo”

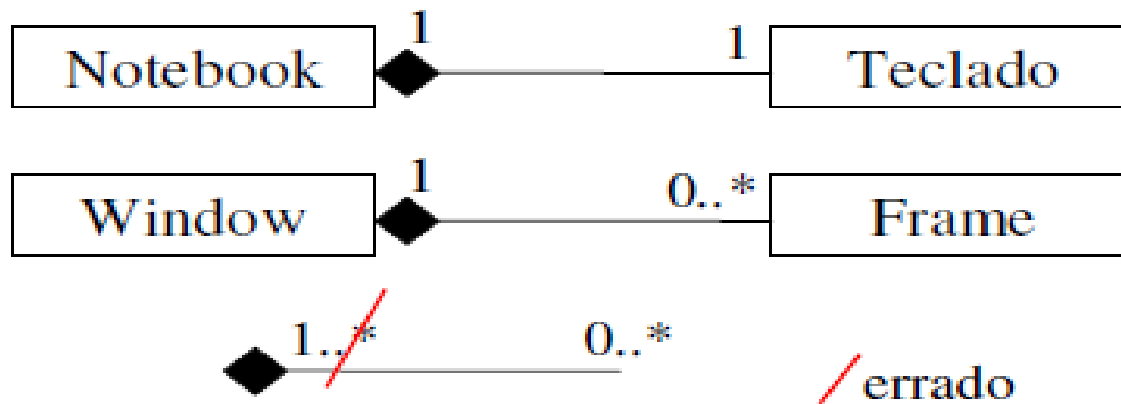


Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Associação de Composição

É uma variante semanticamente mais “forte” da agregação
Os objetos “parte” só podem pertencer a um único objeto “todo” e têm o seu tempo de vida coincidente com o dele



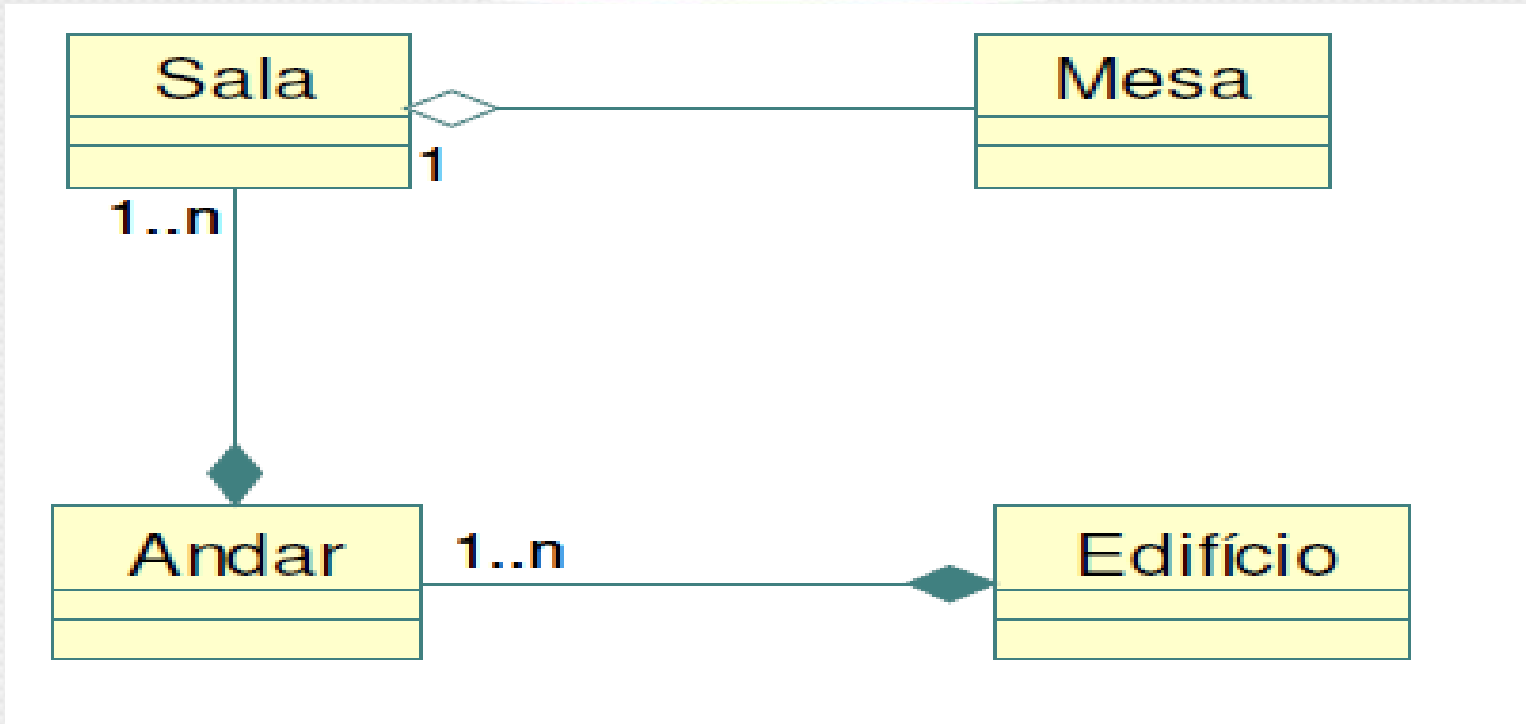
Quando o “todo” *morre* todas as suas “partes” também *morrem*



Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Exemplo: Agregação e composição

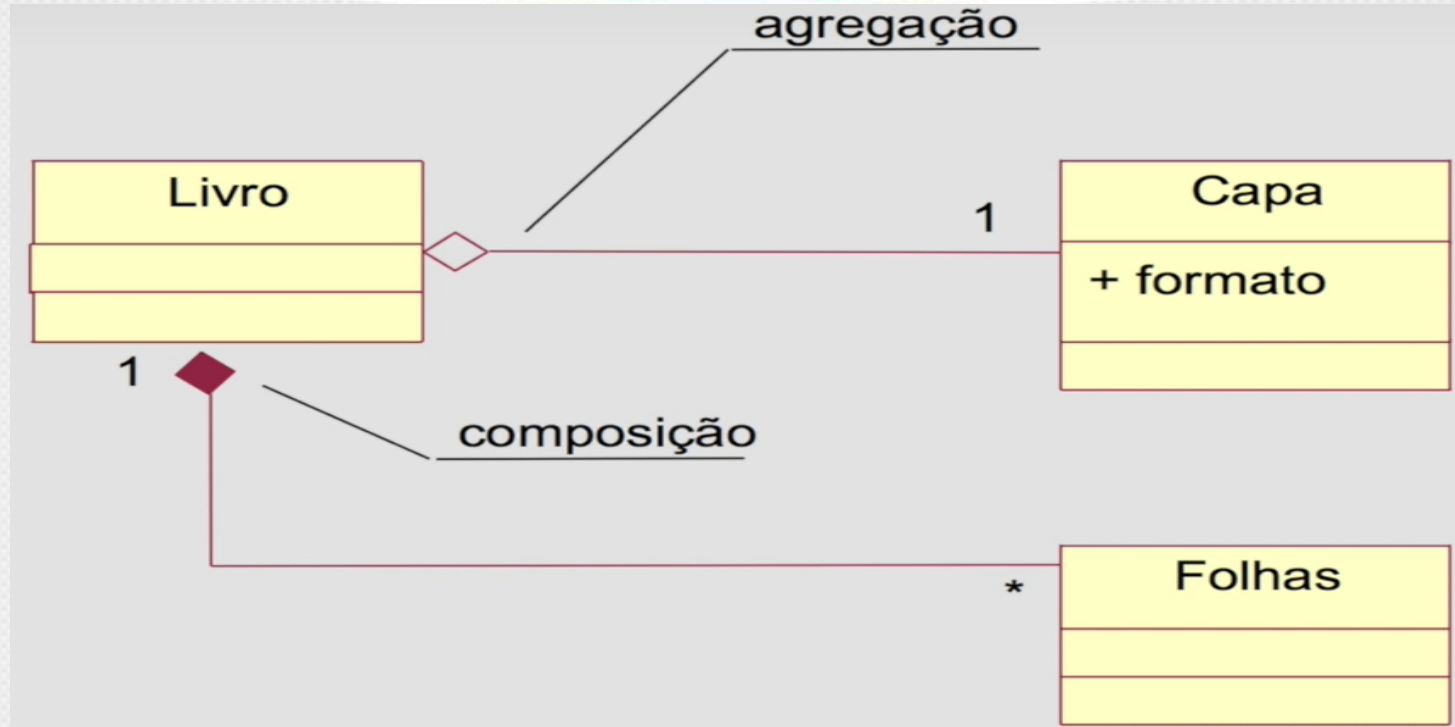


Graduação Tecnológica
Engenharia de Software



UML – Diagrama de classes

Exemplo: Agregação e composição

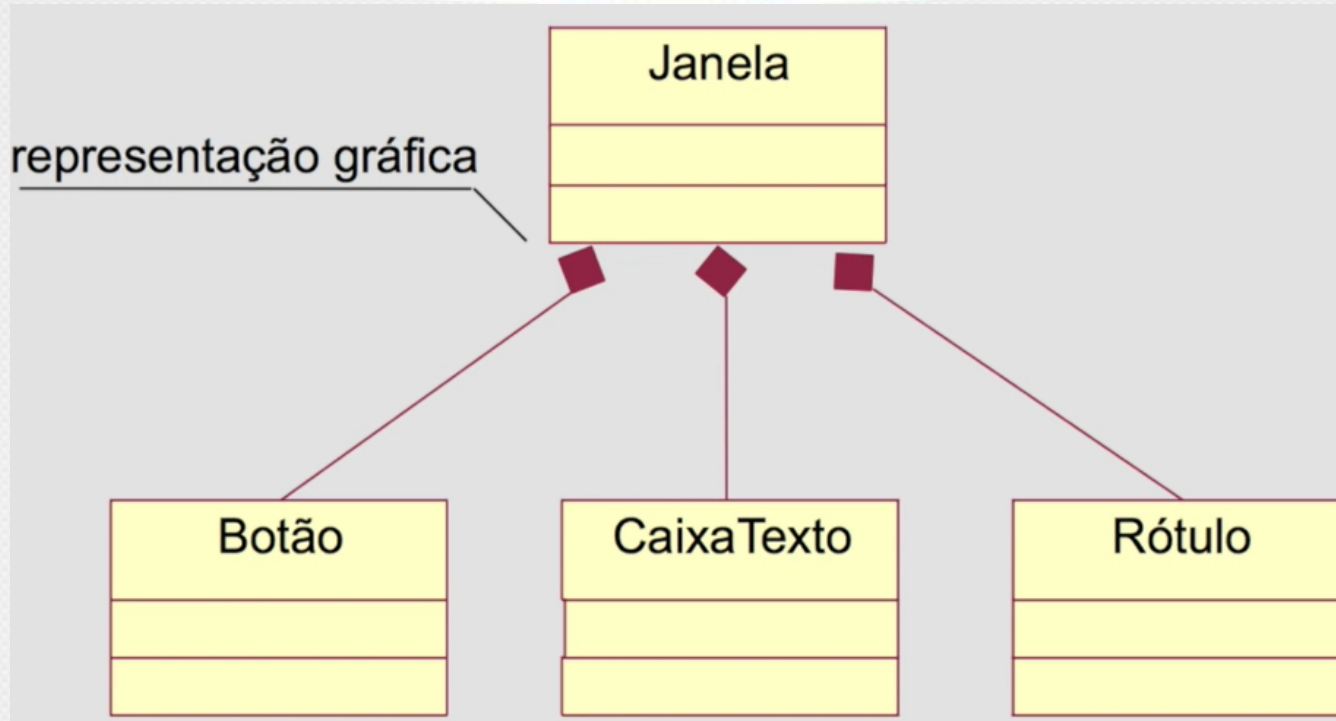


Graduação Tecnológica
Engenharia de Software



UML – Diagrama de classes

Exemplo: Agregação e composição

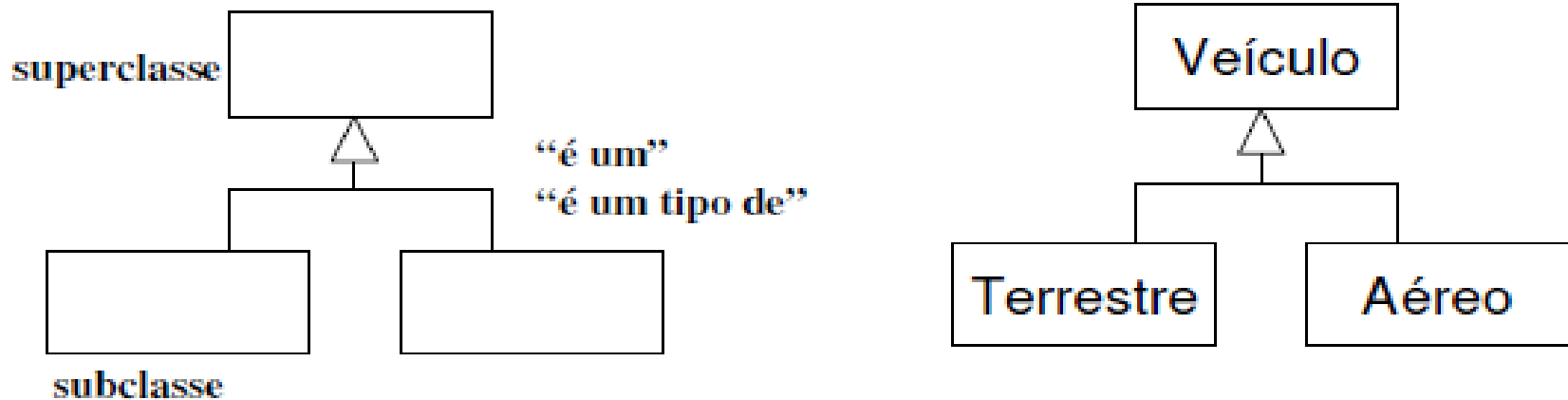


Graduação Tecnológica
Engenharia de Software



UML – Diagrama de classes

Generalização - Herança



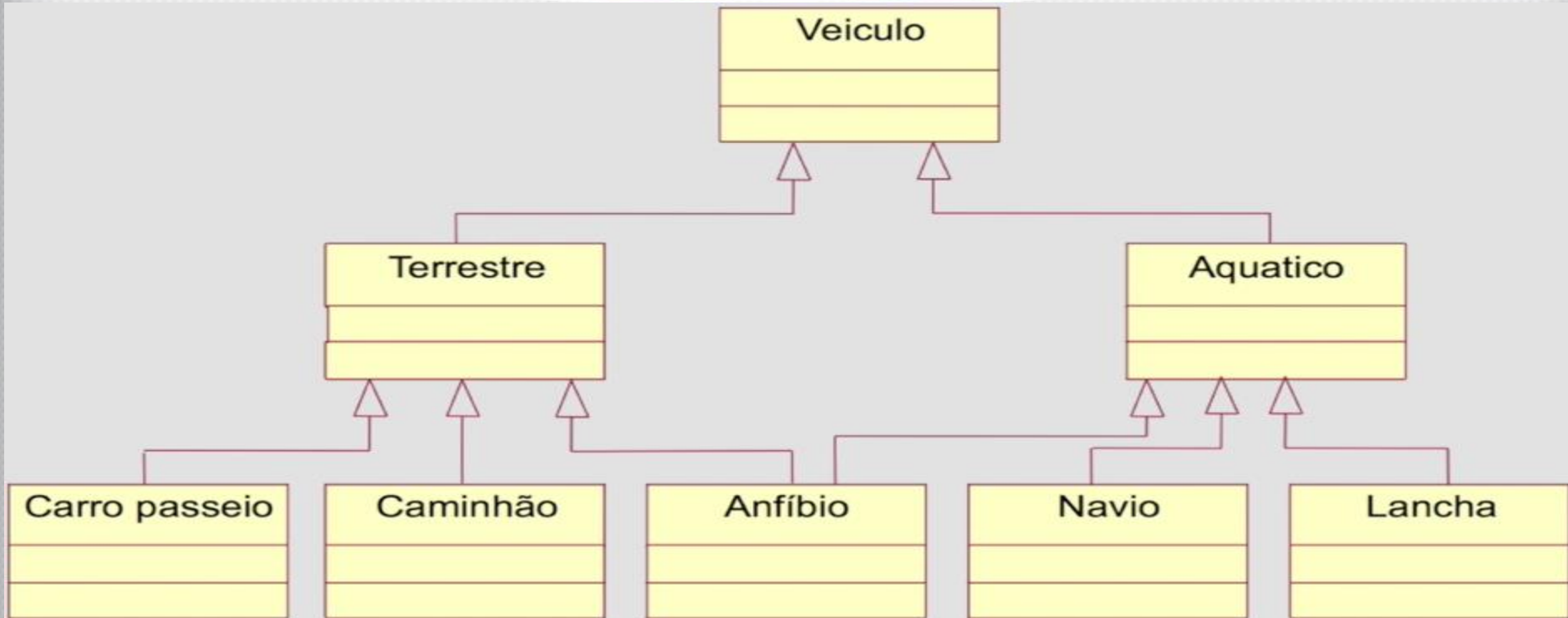
É um relacionamento entre itens gerais (superclasses) e itens mais específicos (subclasses)



Graduação Tecnológica
Engenharia de Software

UML – Diagrama de classes

Generalização - Herança



Graduação Tecnológica
Engenharia de Software



Diagramas de pacote

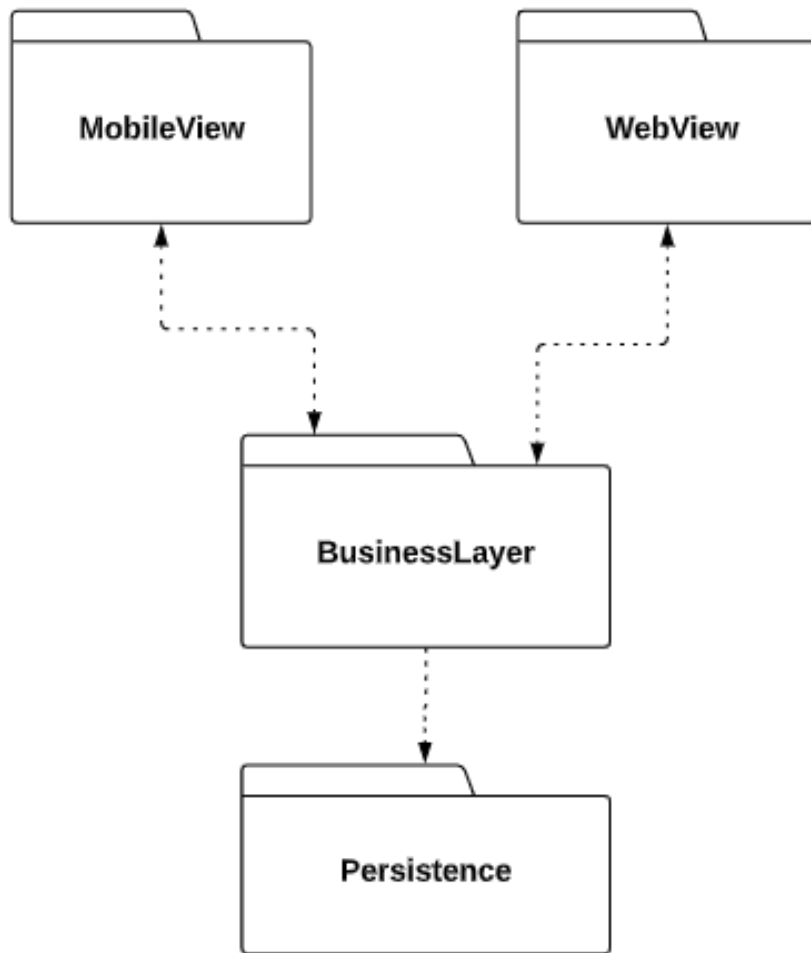


Diagramas de pacotes são recomendados quando se pretende oferecer um modelo de mais alto nível de um sistema, que mostre apenas grupos de classes — isto é, pacotes — e as dependências entre eles.



Graduação Tecnológica
Engenharia de Software

Diagramas de pacote



Em diagramas de pacotes, temos um único tipo de seta, sempre tracejada, que representa qualquer tipo de relacionamento, seja ele por meio de associação, herança ou dependência simples.

Graduação Tecnológica
Engenharia de Software

Diagramas de Sequência

Eles modelam objetos de um sistema

Representam o programa em um determinado cenário de uso

São usados quando se pretende explicar o comportamento de um sistema

Incluem informações sobre quais métodos desses objetos são executados

Os objetos são representados por **retângulos**, com o nome dos objetos modelados



Graduação Tecnológica
Engenharia de Software

Diagramas de Sequência

Os objetos ficam na primeira linha do diagrama

A linha vertical abaixo de cada objeto é seu estado:

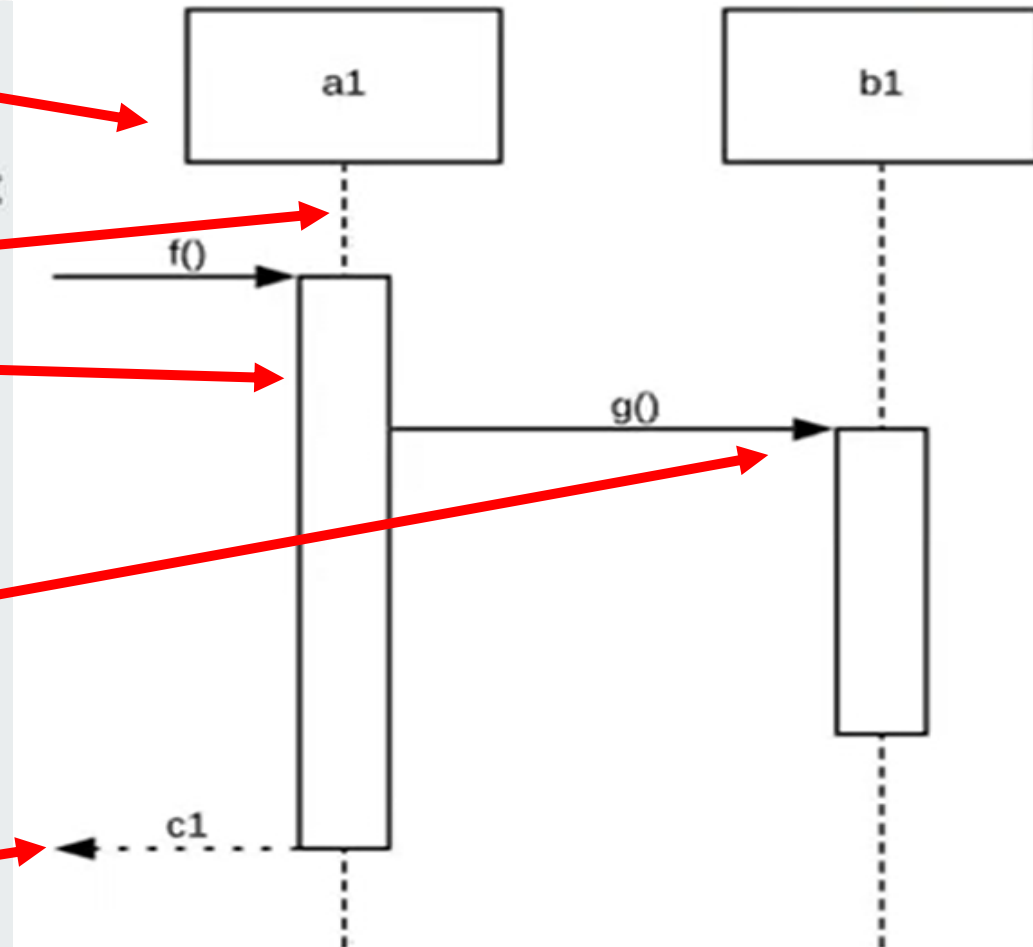
(1) Inativo: linha tracejada

(2) Em execução: forma retangular

O estado do objeto pode alternar diversas vezes

Início da chamada: seta na horizontal + nome do método

Retorno da chamada: seta tracejada + nome do objeto retornado



Diagramas de Sequência

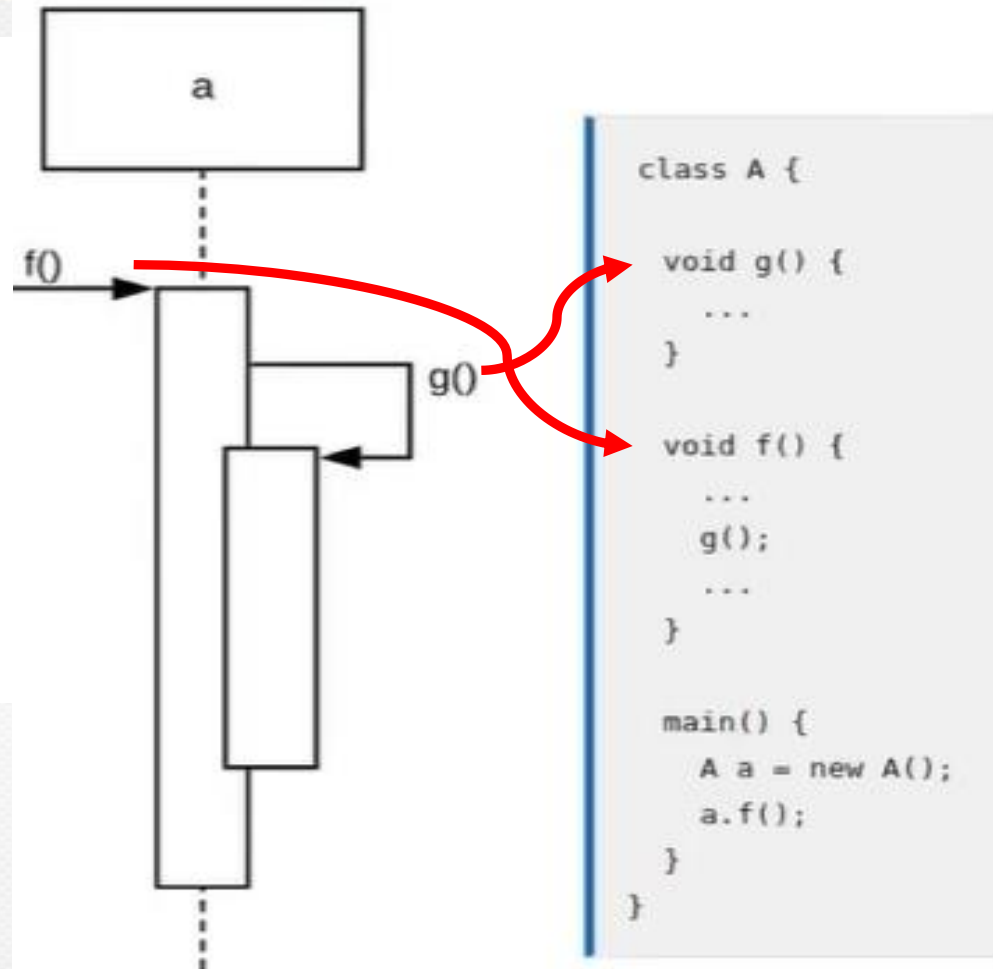
Casos especiais

Caso especial:

Um objeto chama um método dele mesmo

Note que:

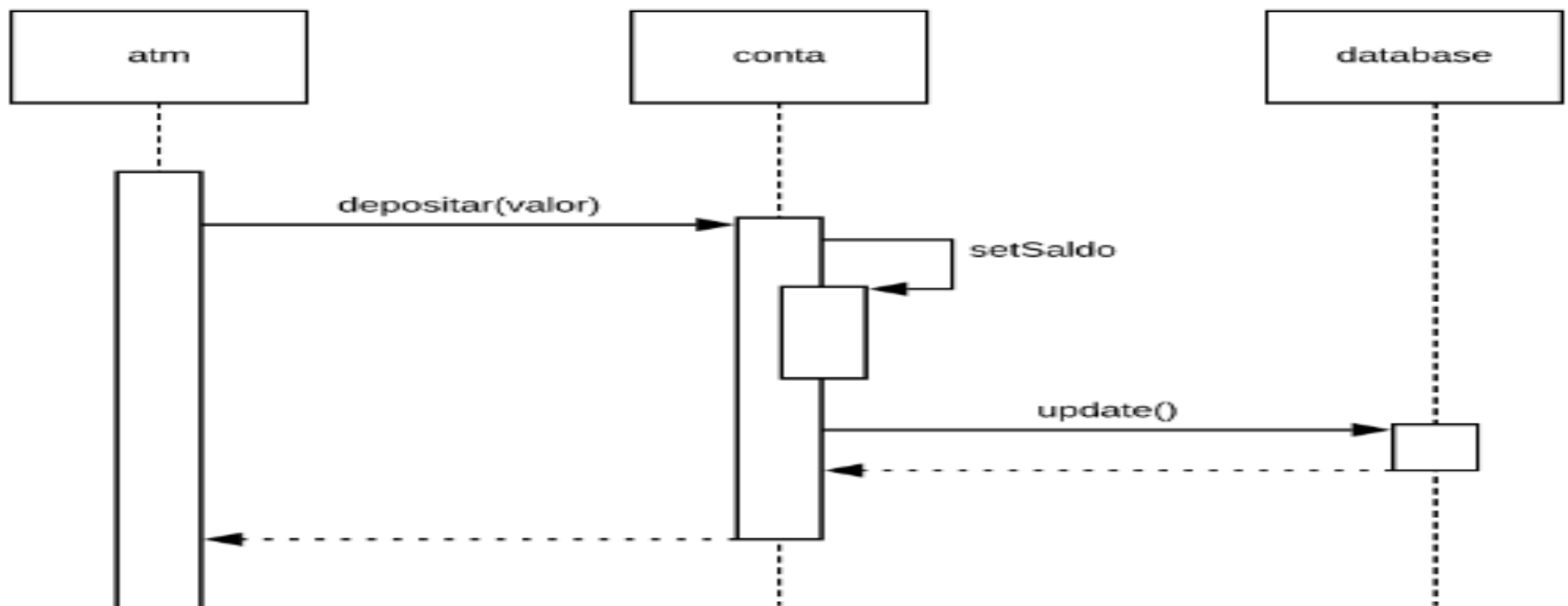
A chamada de g() feita por f() é representada por meio de um novo retângulo, que sai do retângulo que representa a ativação da função f()



Diagramas de Sequência

Exemplo

Nesse exemplo cliente de um caixa eletrônico solicita um depósito de certo valor em sua conta



Diagramas de Atividade

São usados para representar um processo ou fluxo de execução

Seus elementos são:

- **Ações:** retângulos
- **Controle:** ordem de execução das ações

Para entender o funcionamento de um diagrama de atividades:

- Imagine que existe uma ficha imaginária
- Essa ficha caminha pelos elementos do diagrama

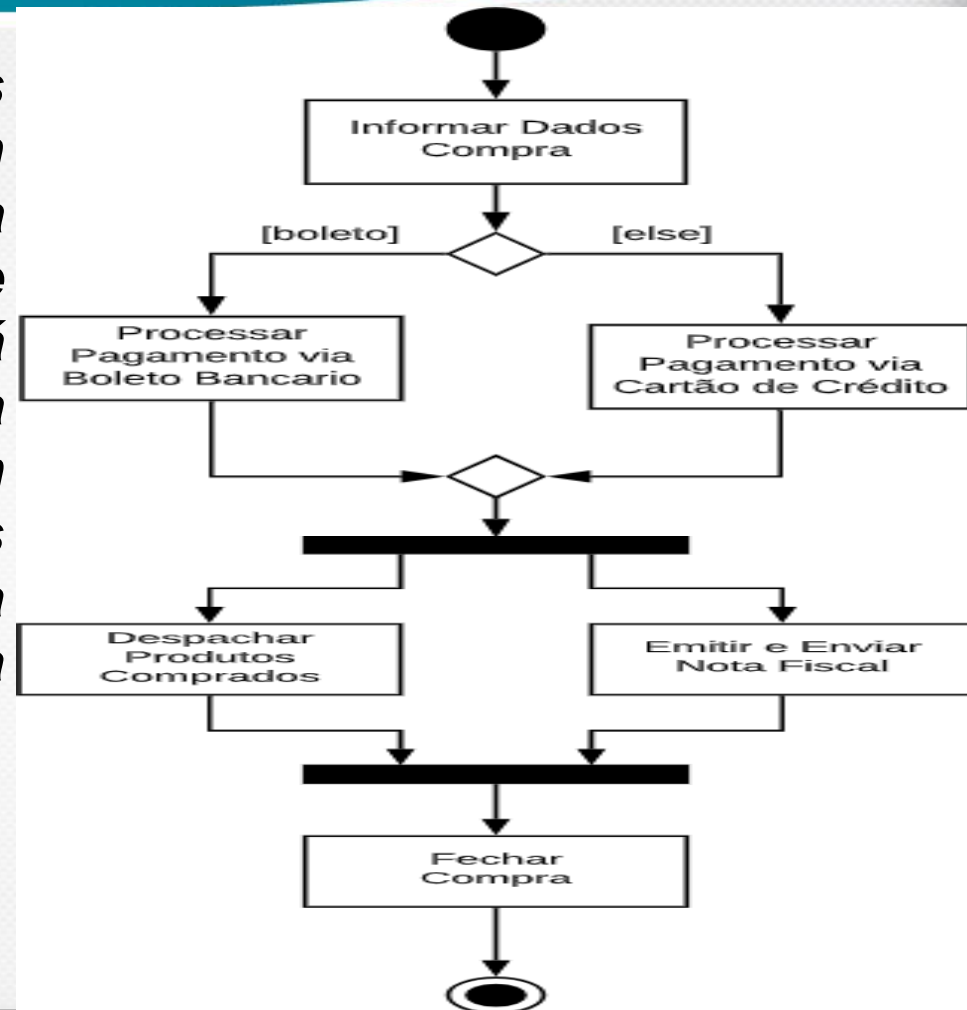


Graduação Tecnológica
Engenharia de Software

Diagramas de Atividade

Exemplo

Exemplo de diagrama de atividades que modela o processo, após um usuário fechar uma compra em uma loja virtual. Para isso, assume-se que os produtos comprados já estão no carrinho de compra. Para entender o funcionamento de um diagrama de atividades, devemos assumir que existe uma ficha (token) imaginária que caminha pelos nodos do diagrama.

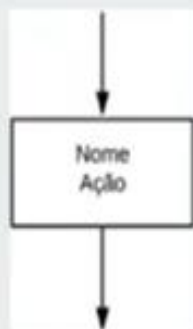


Diagramas de Atividade



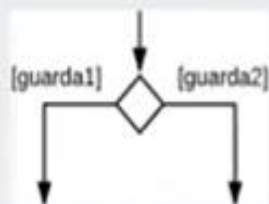
Nodo Inicial

- Não possui fluxo de entrada
- Dá início a execução do processo
- Possui um fluxo de saída



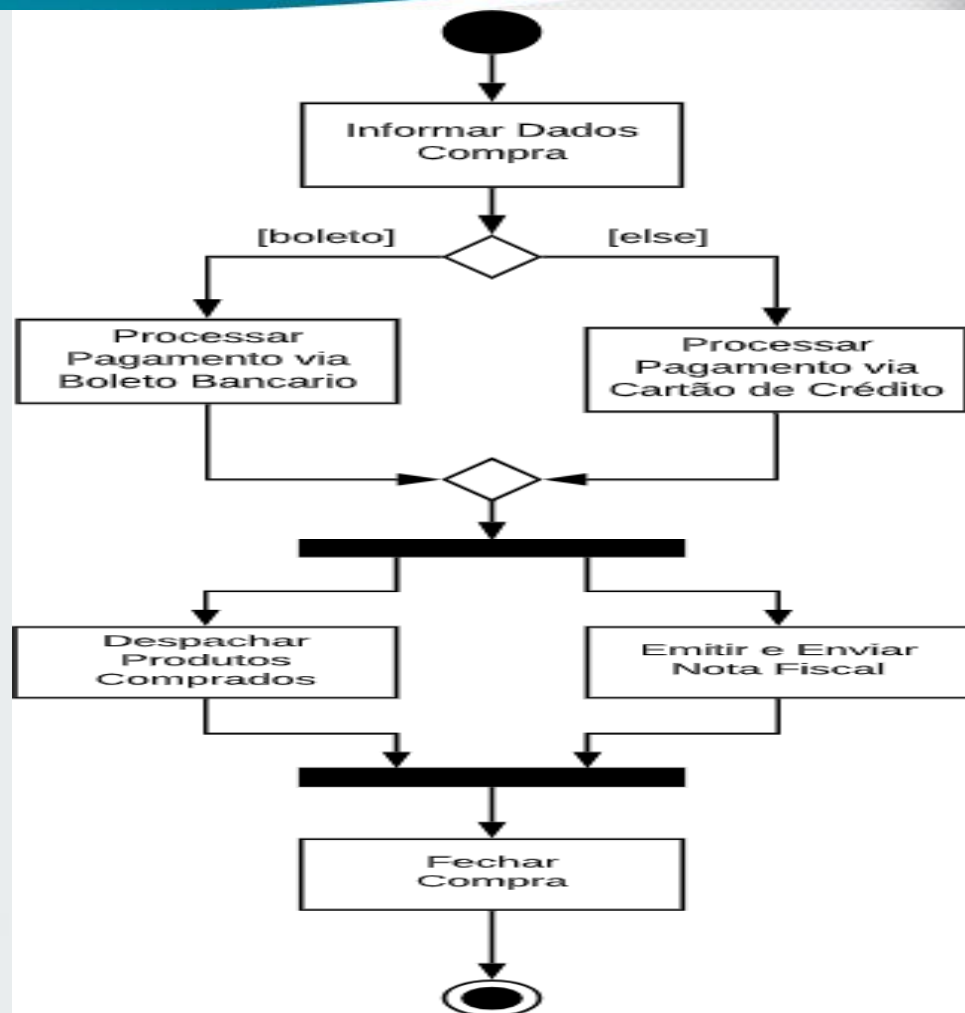
Ações

- Possuem um fluxo de entrada
- Possuem um fluxo de saída



Decisões

- Possuem um fluxo de entrada
- Possuem dois ou mais fluxos de saída
- Apenas um fluxo de saída é escolhido para dar continuidade ao fluxo



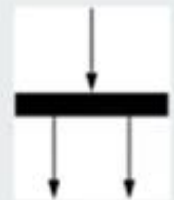
Diagramas de Atividade

Merges



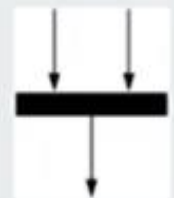
- Podem possuir vários fluxos de entrada
- Possuem um fluxo de saída
- São usados para unir os fluxos de nodos de decisão

Forks



- Possuem um fluxo de entrada
- Possuem um ou mais fluxos de saída
- Criam múltiplos processos em execução de forma paralela

Joins

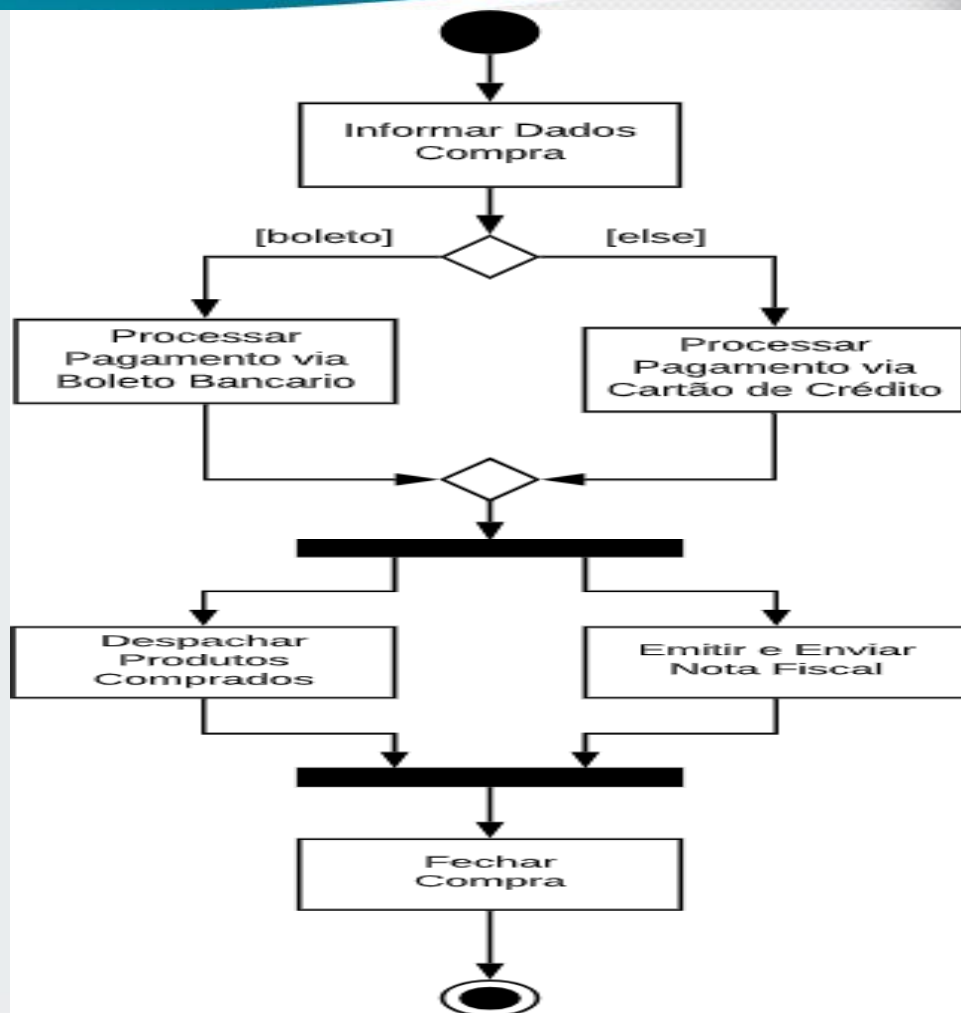


- Possuem vários fluxos de entrada
- Possuem um fluxo de saída
- Transformam vários fluxos de execução em um único fluxo

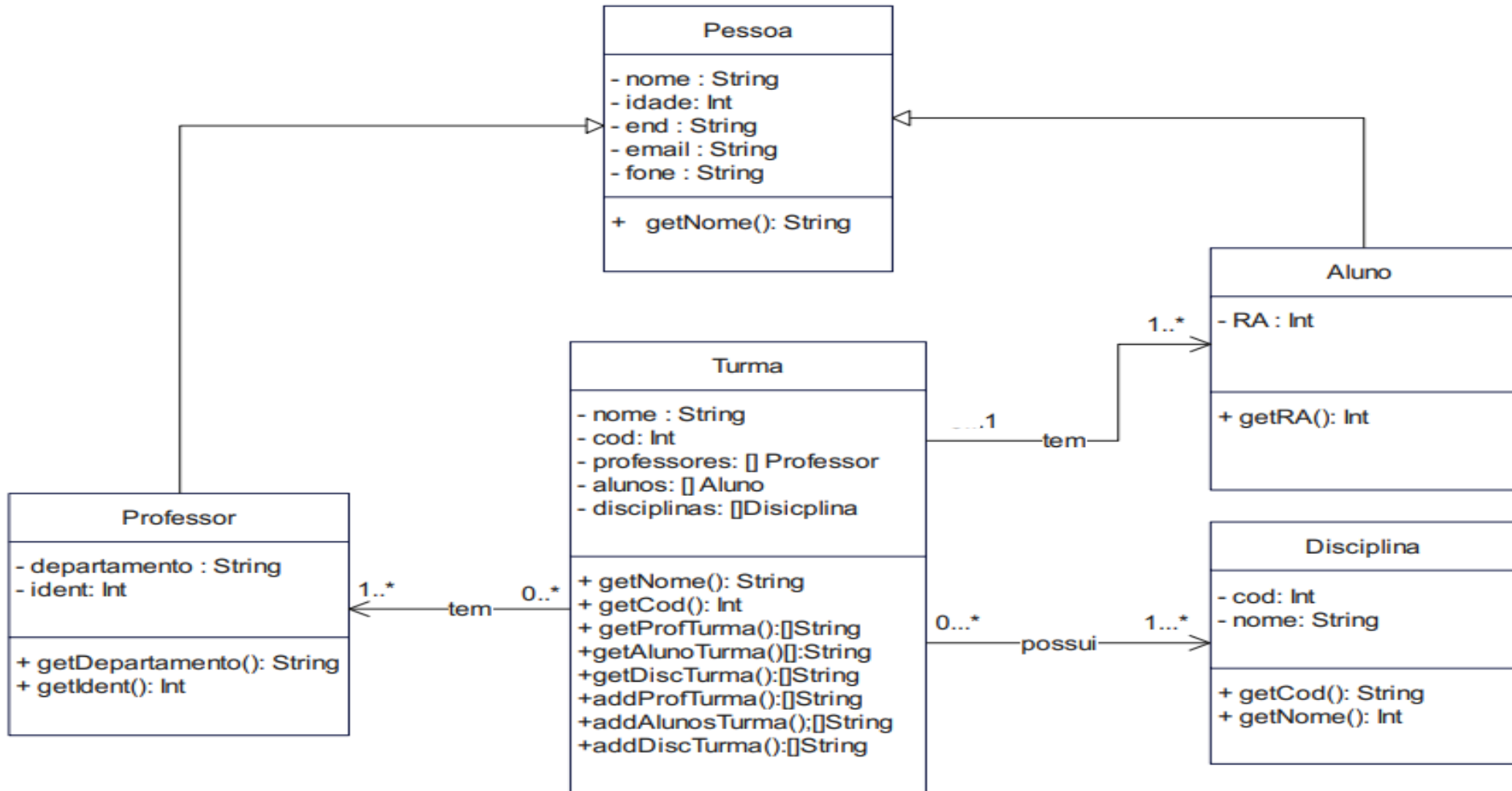
Nodo Final



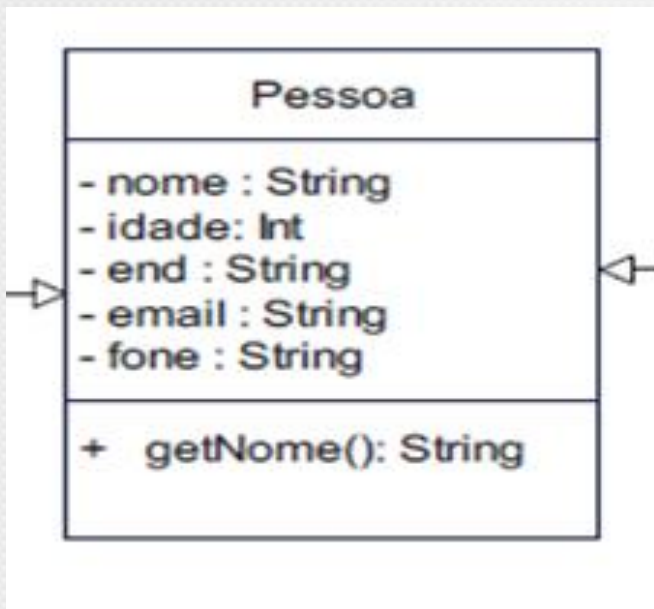
- Não possui fluxos de saídas
- Possui mais de um fluxo de entrada
- Encerra a execução do processo



Exemplo1: Diagrama de Classes



Exemplo1: Código do modelo – classe Pessoa



```
class Pessoa:

    def __init__(self, nome, idade, end, email, fone):

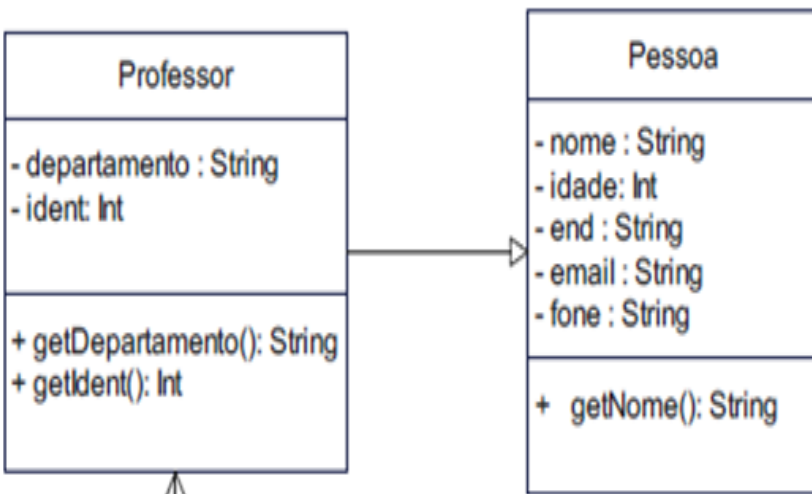
        self.__nome = nome
        self.__idade = idade
        self.__end = end
        self.__email = email
        self.__fone = fone

    def getNome(self):
        return self.__nome
```

****Crie os outros métodos, usando o método getNome() como exemplo***



Exemplo1: Código do modelo – classe Professor



```
class Professor(Pessoa):
```

```
    def __init__(self, nome, idade, end, email, fone, departamento, ident):
```

```
        super().__init__(nome, idade, end, email, fone)
```

```
        self.__departamento = departamento
```

```
        self.__ident = ident
```

```
    def getDepartamento(self):
```

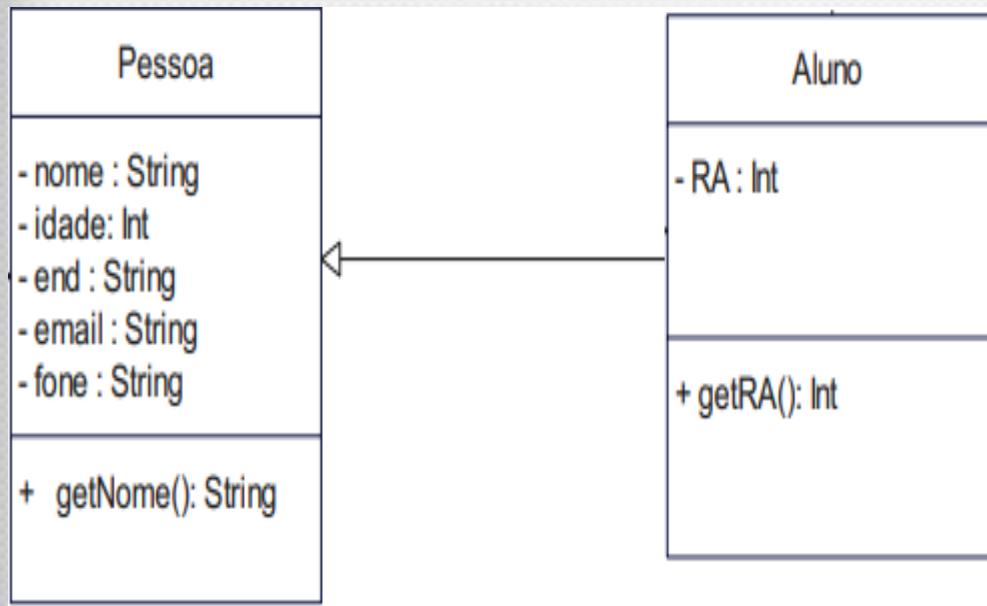
```
        return self.__departamento
```

```
    def getIdent(self):
```

```
        return self.__ident
```



Exemplo1: Código do modelo – classe Aluno



****Crie o código da herança conforme o modelo apresentado para Professor***



Exemplo1: Código do modelo – Classe Disciplina

Disciplina
- cod: Int - nome: String
+ getCod(): String + getNome(): Int

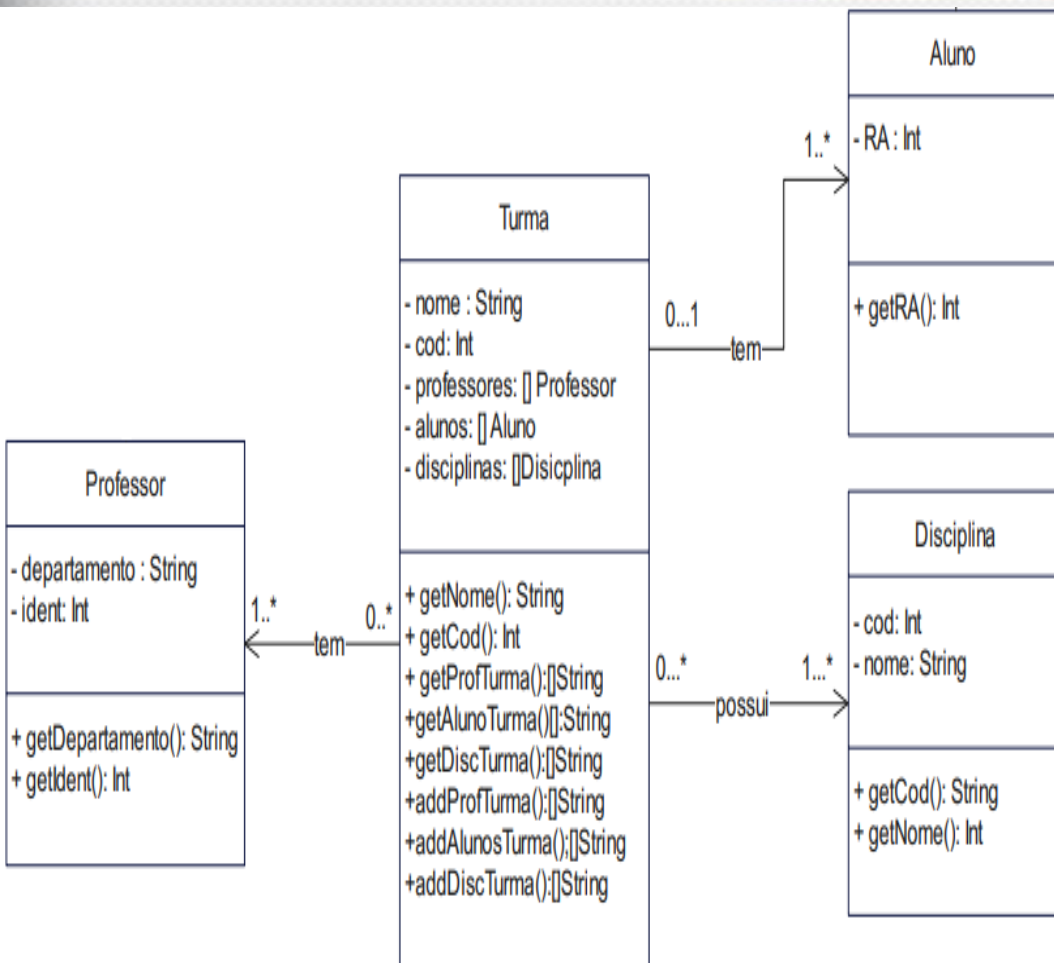
```
class Disciplina:
    def __init__(self, cod, nome):
        self.__cod = cod
        self.__nome = nome

    def getCod(self):
        return self.__cod

    def getNome(self):
        return self.__nome
```



Exemplo1: Código do modelo – Classe Turma



```
class Turma:
```

```
def __init__(self, nome, cod):
```

```
    self.__nome = nome
```

```
    self.__cod = cod
```

```
    self.__professores = []
```

```
    self.__alunos = []
```

```
    self.__disciplinas = []
```

```
def getNome(self):
```

```
    return self.__nome
```

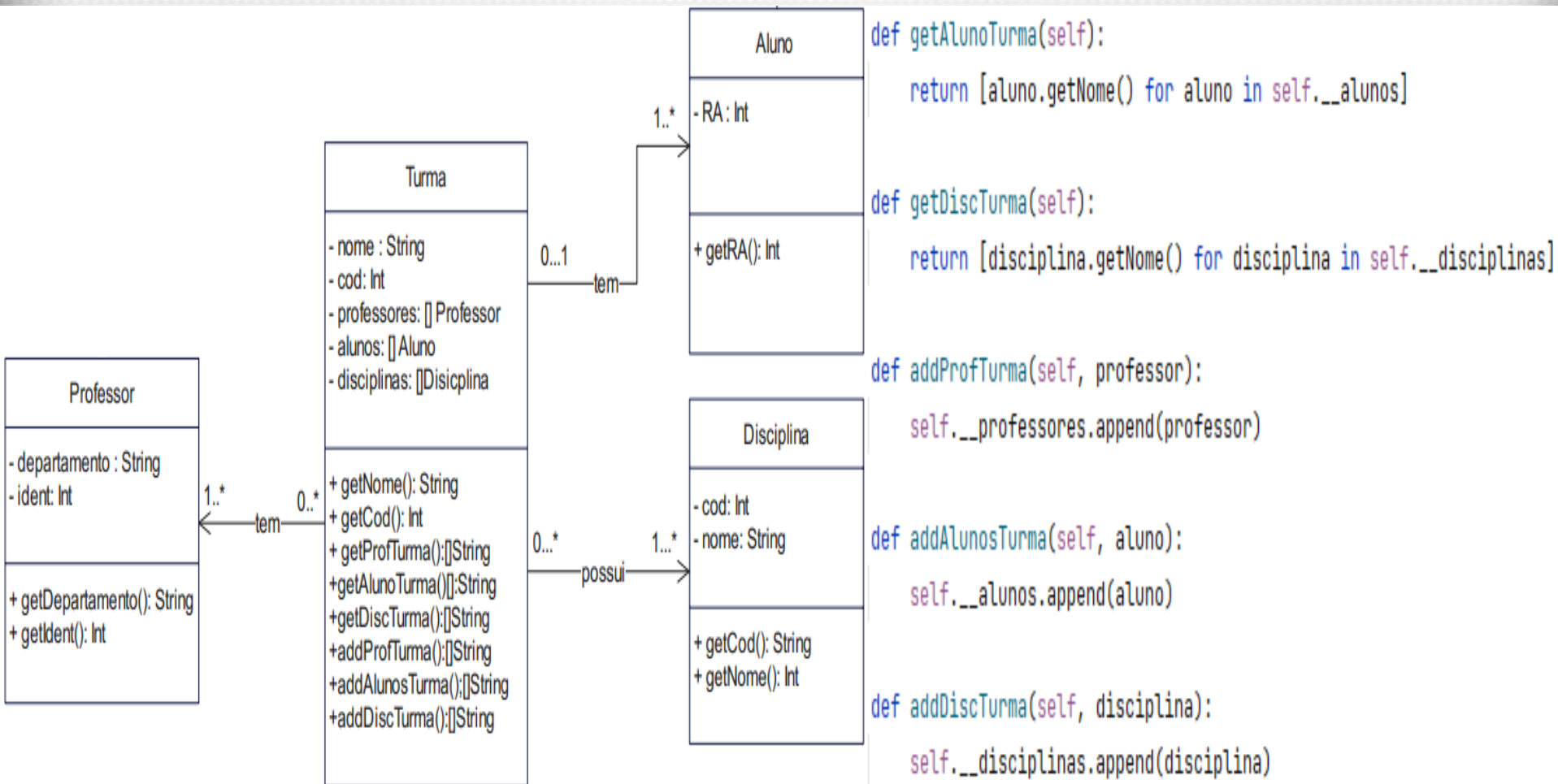
```
def getCod(self):
```

```
    return self.__cod
```

```
def getProfTurma(self):
```

```
    return [professor.getNome() for professor in self.__professores]
```

Exemplo1: Código do modelo – Classe Turma



Exemplo1: Testando o código

```
# Exemplo de aplicação
if __name__ == "__main__":
    # Criando um professor
    professor1 = Professor("Luciano Santos", 45, "Rua A, 123", "luciano@exemplo.com", "1234-5678", "ADS", 1001)

    # Criando alunos
    aluno1 = Aluno("Sicrana", 20, "Rua B, 456", "sicrana@exemplo.com", "9876-5432", 2001)
    aluno2 = Aluno("Fulano", 21, "Rua C, 789", "fulano@exemplo.com", "6543-2109", 2002)

    # Criando disciplinas
    disciplina1 = Disciplina(101, "Linguagem de Programação")
    disciplina2 = Disciplina(102, "Engenharia de Software")

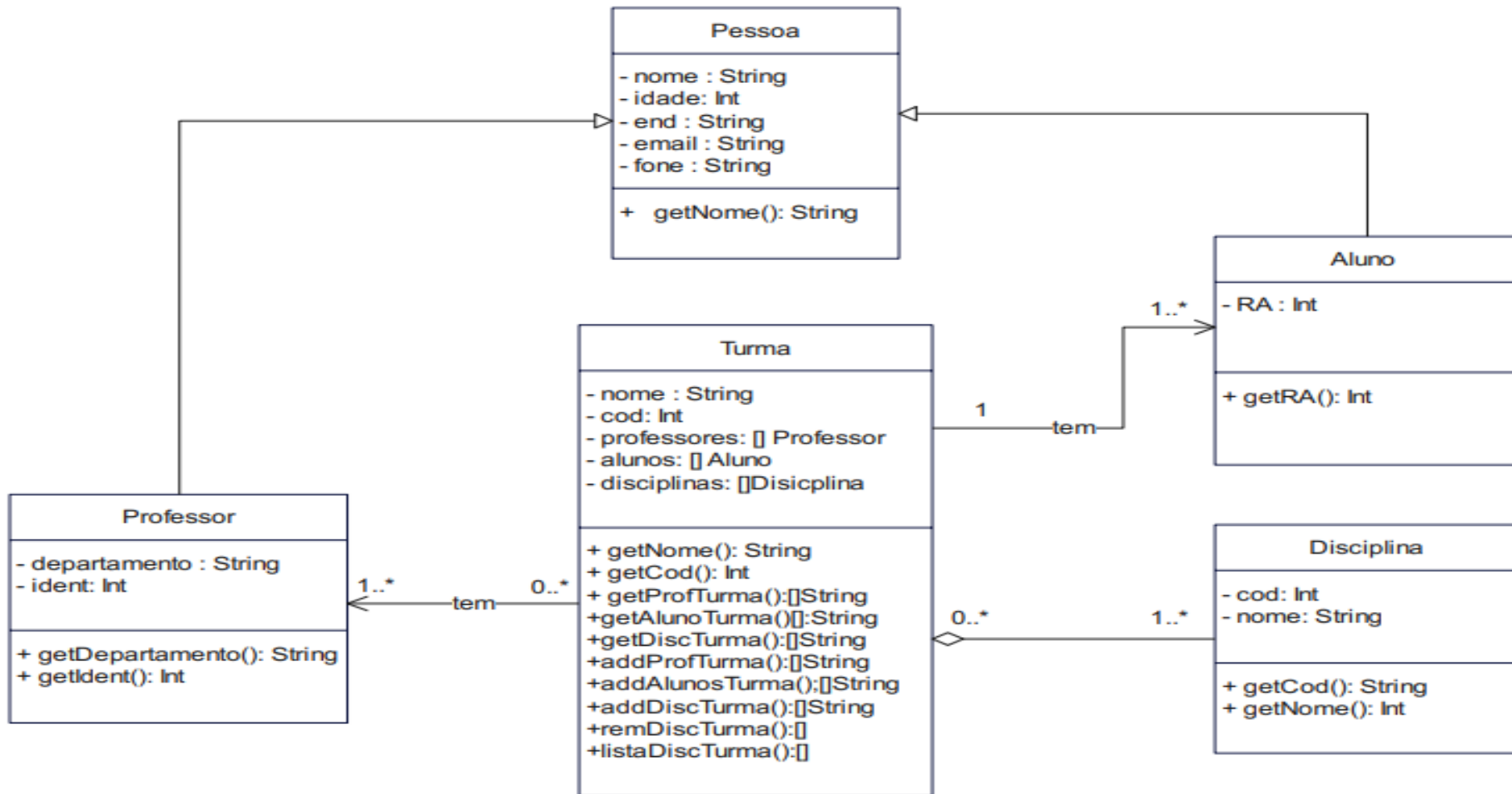
    # Criando uma turma
    turma = Turma("ADS_2_2024", 2024)
```


Exemplo1: Testando o código

```
# Adicionando professor, alunos e disciplinas à turma
turma.addProfTurma(professor1)
turma.addAlunosTurma(aluno1)
turma.addAlunosTurma(aluno2)
turma.addDiscTurma(disciplina1)
turma.addDiscTurma(disciplina2)

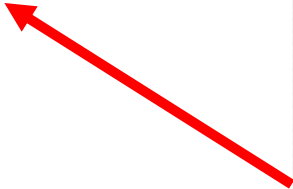
# Exibindo informações da turma
print("Professores da turma:", turma.getProfTurma())
print("Alunos da turma:", turma.getAlunoTurma())
print("Disciplinas da turma:", turma.getDiscTurma())
```

Exemplo2: Lançando uma associação de agregação



Exemplo2: Alterar a classe Turma

```
def addDiscTurma(self, disciplina):  
    if isinstance(disciplina, Disciplina):  
        self.__disciplinas.append(disciplina)  
  
def remDiscTurma(self, disciplina):  
    if disciplina in self.__disciplinas:  
        self.__disciplinas.remove(disciplina)  
  
def listarDiscTurma(self):  
    print(f"Turma {self.__cod} tem as seguintes disciplinas:")  
    for disciplina in self.__disciplinas:  
        print(f"- {disciplina.getNome()}")
```

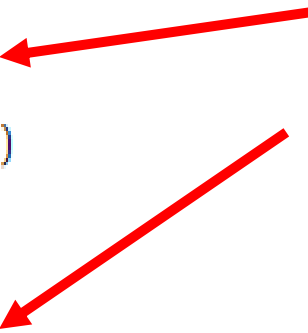


****Crie um código novo e altere na classe Turma o método addDiscTurma() do exemplo anterior.***

Exemplo2: Alterar a classe Turma

```
def addDiscTurma(self, disciplina):  
    if isinstance(disciplina, Disciplina):  
        self.__disciplinas.append(disciplina)  
  
def remDiscTurma(self, disciplina):  
    if disciplina in self.__disciplinas:  
        self.__disciplinas.remove(disciplina)  
  
def listarDiscTurma(self):  
    print(f"Turma {self.__cod} tem as seguintes disciplinas:")  
    for disciplina in self.__disciplinas:  
        print(f"- {disciplina.getNome()}")
```

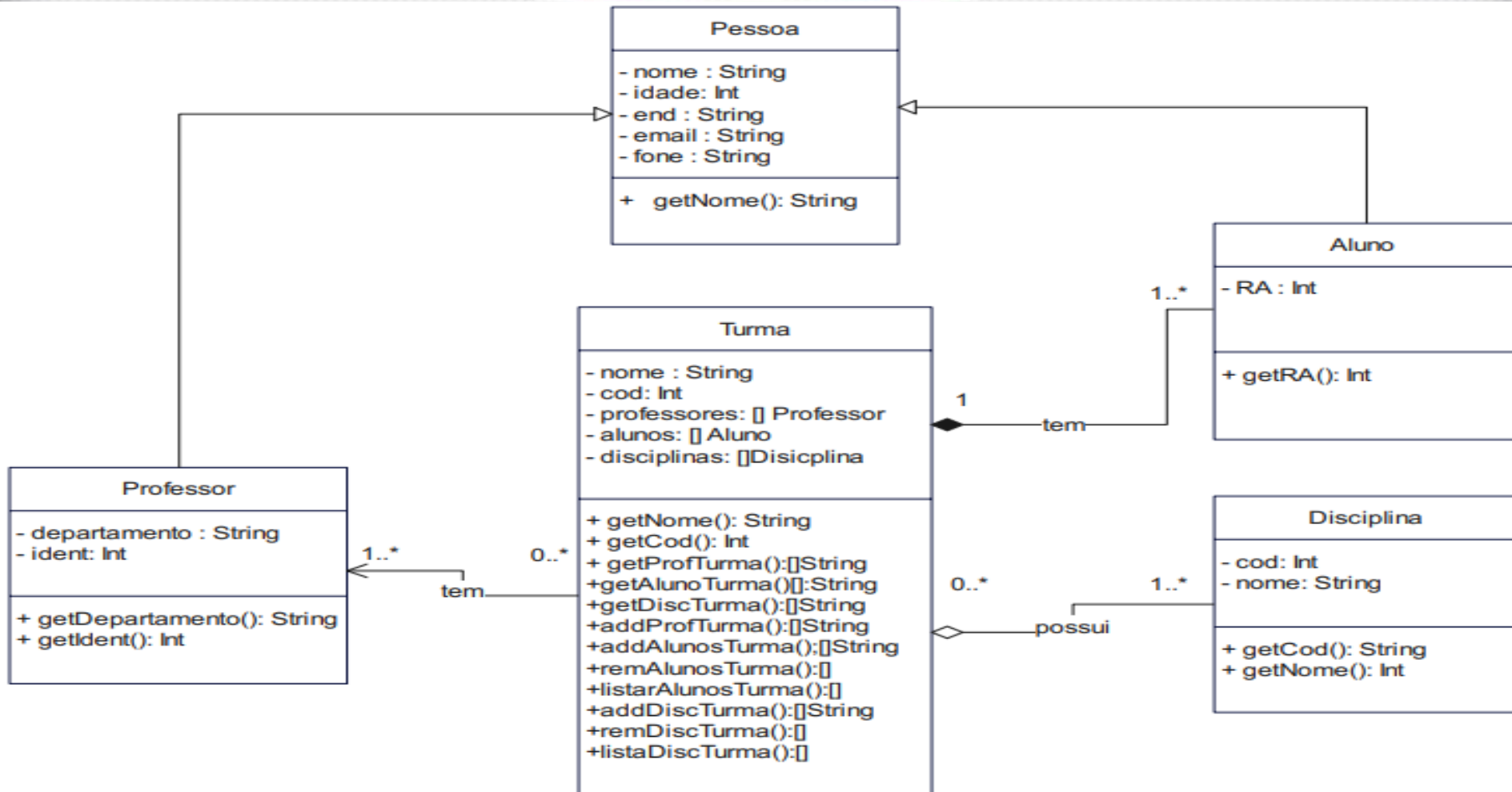
****Acréscete aos métodos já existentes, os métodos de remoção e listagem para a classe Turma***



Exemplo2: Teste o código

```
# Exemplo de aplicação
if __name__ == "__main__":
    # Criando disciplinas
    disciplina1 = Disciplina(101, "Linguagem de Programação")
    disciplina2 = Disciplina(102, "Engenharia de Software")
    # Criando uma turma
    turma = Turma("ADS_2_2024", 2024)
    # Agregando disciplinas à turma
    turma.addDiscTurma(disciplina1)
    turma.addDiscTurma(disciplina2)
    # Listando disciplinas da turma
    turma.listarDiscTurma()
    # Removendo uma disciplina da turma
    turma.remDiscTurma(disciplina1)
    turma.listarDiscTurma()
    # Mostrando que a disciplina ainda existe após ser removida da turma
    print(f"A disciplina {disciplina1.getNome()} ainda existe independentemente da turma.")
    print(f"A disciplina {disciplina2.getNome()} ainda existe independentemente da turma.")
```

Exemplo3: Lançando uma associação de composição



Exemplo3: Alterar novamente a classe Turma

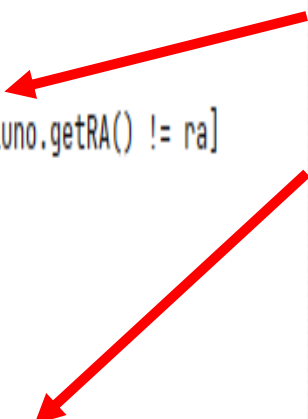
```
def addAlunosTurma(self, nome, idade, end, email, fone, ra):  
    aluno = Aluno(nome, idade, end, email, fone, ra) # Cria um novo objeto Aluno  
    self.__alunos.append(aluno)  
  
def remAlunosTurma(self, ra):  
    self.__alunos = [aluno for aluno in self.__alunos if aluno.getRA() != ra]  
    # Remove o aluno pela matrícula  
  
def listarAlunosTurma(self):  
    print(f"Turma {self.__cod} tem os seguintes alunos:")  
    for aluno in self.__alunos:  
        print(f"- {aluno.getNome()} (Matrícula: {aluno.getRA()}")
```

***Crie um código novo e altere na classe Turma o método addAlunosTurma() dos exemplos anteriores. Notar que a instância da criação do objeto deve ficar dentro da classe primária, que se destruída, também destrói o objeto secundário**

Exemplo3: Alterar novamente a classe Turma

```
def addAlunosTurma(self, nome, idade, end, email, fone, ra):  
    aluno = Aluno(nome, idade, end, email, fone, ra) # Cria um novo objeto Aluno  
    self.__alunos.append(aluno)  
  
def remAlunosTurma(self, ra):  
    self.__alunos = [aluno for aluno in self.__alunos if aluno.getRA() != ra]  
    # Remove o aluno pela matrícula  
  
def listarAlunosTurma(self):  
    print(f"Turma {self.__cod} tem os seguintes alunos:")  
    for aluno in self.__alunos:  
        print(f"- {aluno.getNome()} (Matrícula: {aluno.getRA()}")
```

***Crie dois novos métodos aos já existentes, um para remover alunos e outro para listar os alunos da turma.**



Exemplo3: Teste o código

```
# Exemplo de aplicação
if __name__ == "__main__":
    # Criando uma turma
    turma = Turma("ADS_2_2024", 2024)
    # Adicionando alunos à turma
    turma.addAlunosTurma("Sicrana", 20, "Rua B, 456", "sicrana@exemplo.com", "9876-5432", 2001)
    turma.addAlunosTurma("Fulano", 21, "Rua C, 789", "fulano@exemplo.com", "6543-2109", 2002)
    # Listando alunos da turma
    turma.listarAlunosTurma()
    # Removendo um aluno da turma
    turma.remAlunosTurma(2001)
    turma.remAlunosTurma(2002)
    # Listando alunos da turma
    turma.listarAlunosTurma()
    # Adicionando alunos à turma
    turma.addAlunosTurma("Sicrana", 20, "Rua B, 456", "sicrana@exemplo.com", "9876-5432", 2001)
    # Listando alunos da turma
    turma.listarAlunosTurma()
    # Note que, se a turma for destruída, todos os alunos também serão destruídos
    del turma # Além de destruir a instância turma
               # também destruirá todos os objetos Alunos associados a essa turma
```

Exercícios diagramas de classe

Para os exercícios a seguir criar o diagrama de classes com seus atributos e seus métodos mínimos necessários e fazer o relacionamento entre as classes. Usar o encapsulamento privado para os atributos e públicos para os métodos de acesso, multiplicidade para identificação para o tipo de dados.

A partir do diagrama de classe deverá ser implementado a codificação do mesmo e ser realizado um teste da aplicação.



Exercício 1: Sistema de Gestão de Bibliotecas

****Usar Herança e associação simples:***

ItemBiblioteca (superclasse): atributos → código e título

ItemBiblioteca(superclasse): métodos comuns da classe

Livro(subclasse): atributos → autor e gênero

Livro(subclasse): métodos comuns da classe

Revista(subclasse): atributos → edição, mêsPublicação

Revista(subclasse): métodos comuns da classe

Biblioteca(associação simples): atributos → nome, itens

Biblioteca: métodos → adicionarItem, removerItem, listarItens e comuns da classe



Exercício 2: Sistema de controle de pedidos de um Restaurante

****Usar agregação e composição:***

Prato: atributos → nome e preço

Prato: métodos comuns da classe

Pedido: atributos → numPedido, pratos

Pedido: addPrato, remPrato, calcularTotal, comuns da classe

Cliente: atributos → nome, pedidos

Cliente: métodos fazerPedido, comuns da classe



Exercício 3: Sistema de Controle de Veículos para um motorista

****Usar herança e composição:***

Veículo: atributos → placa, marca, tipo

Veículo: métodos comuns da classe

Carro: atributos → numportas

Carro: métodos comuns da classe

Caminhão: atributos → capacidadeCarga

Cliente: métodos comuns da classe

Motorista: nome, veículo

Motorista: métodos → setVeiculo, comuns da classe

