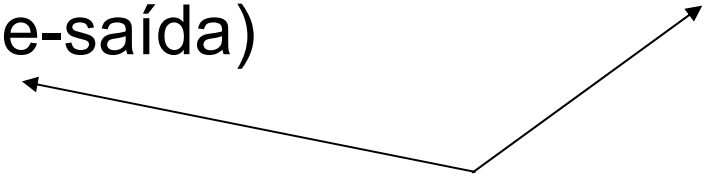
Conceitos/princípios da orientação a objeto

Detalhamento de uma mensagem:

objeto-alvo.nome-do-método(argumentos-de-entrada;
argumentos-de-saída)

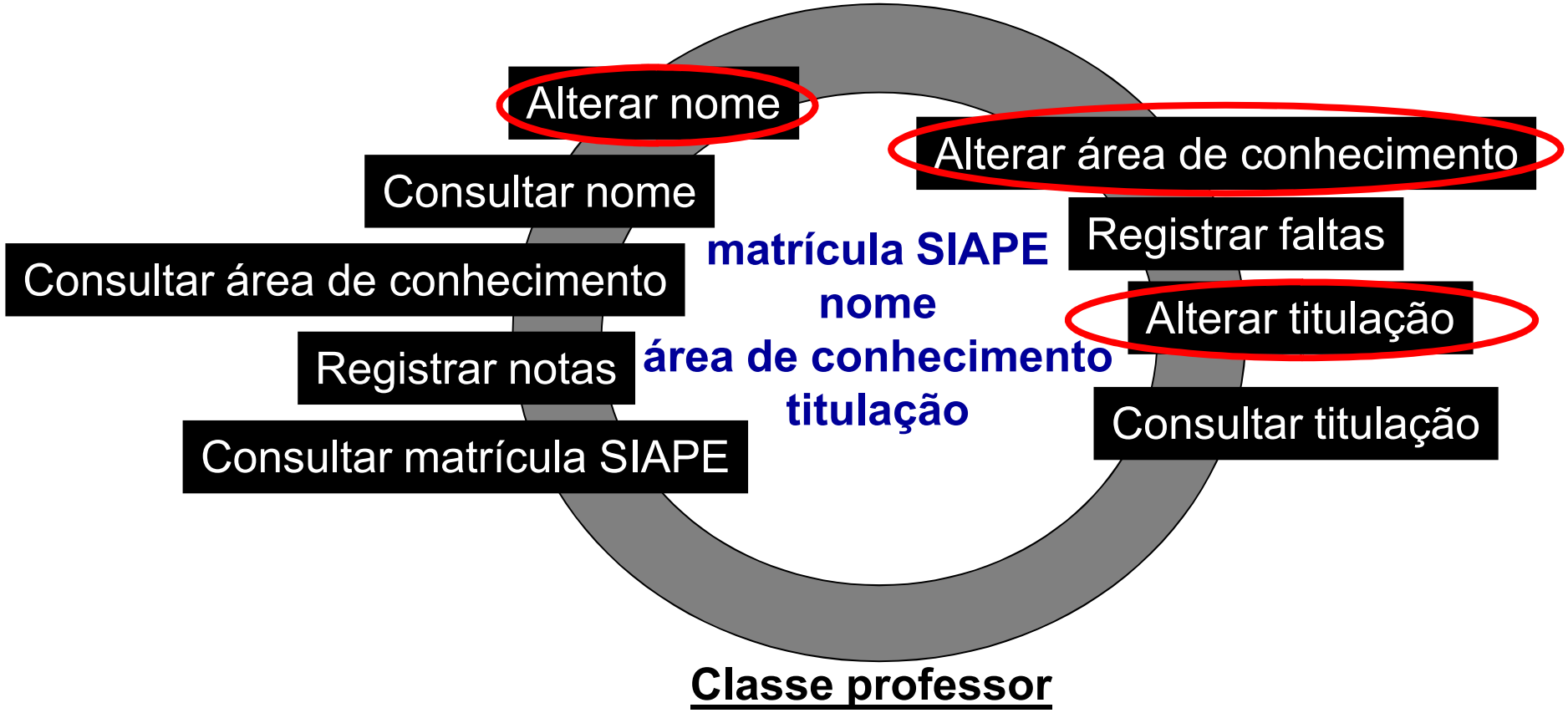


assinatura

Tipos de mensagem:

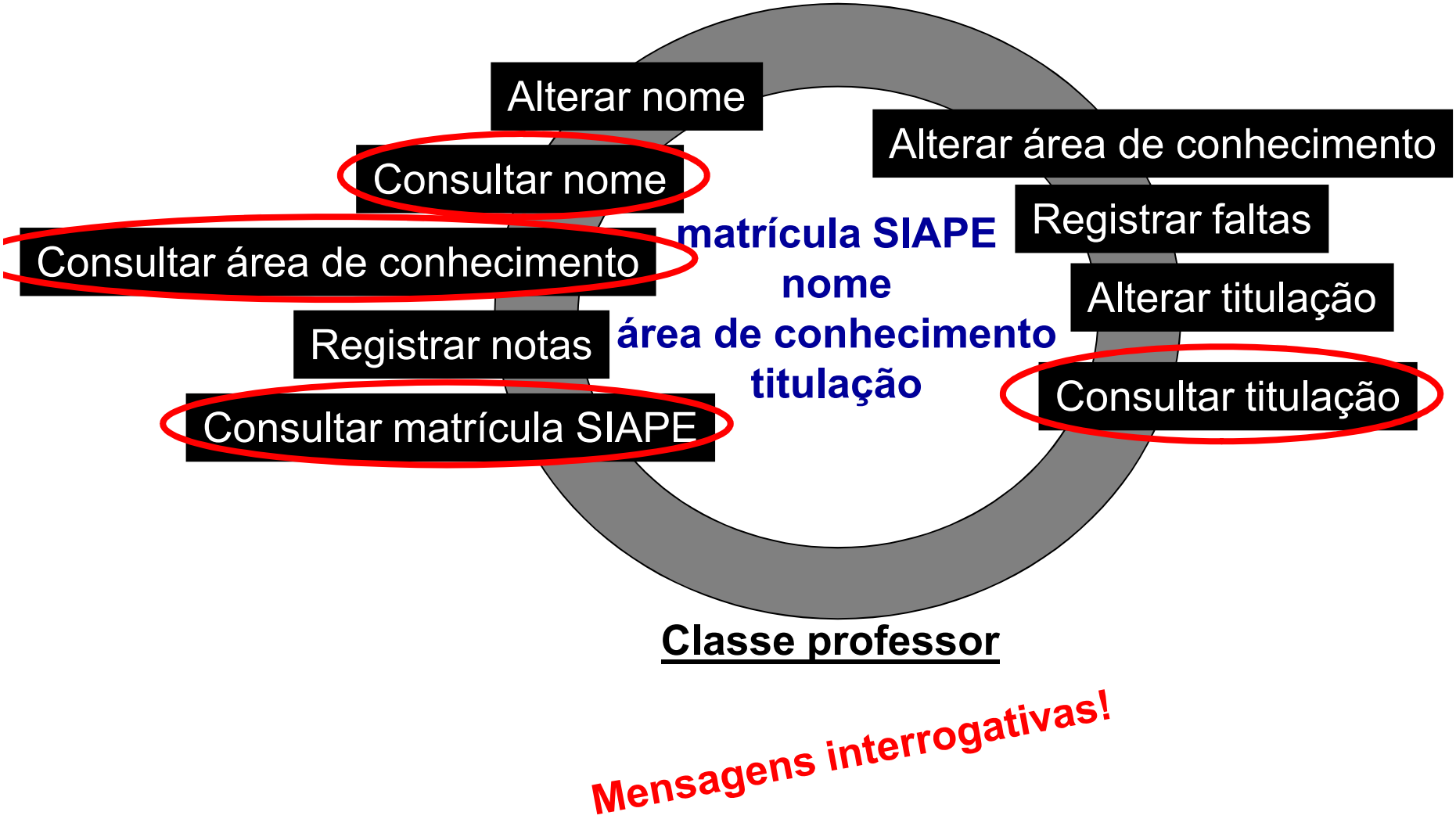
- Mensagem informativa -> fornece informações para que o objeto-alvo atualize seu estado;
- Mensagem interrogativa -> solicita informações a respeito do estado do objeto-alvo;
- Mensagem imperativa -> solicita que o objeto-alvo faça alguma ação sobre si, sobre outro objeto ou sobre o ambiente/sistema em que este se encontra.

Conceitos/princípios da orientação a objeto



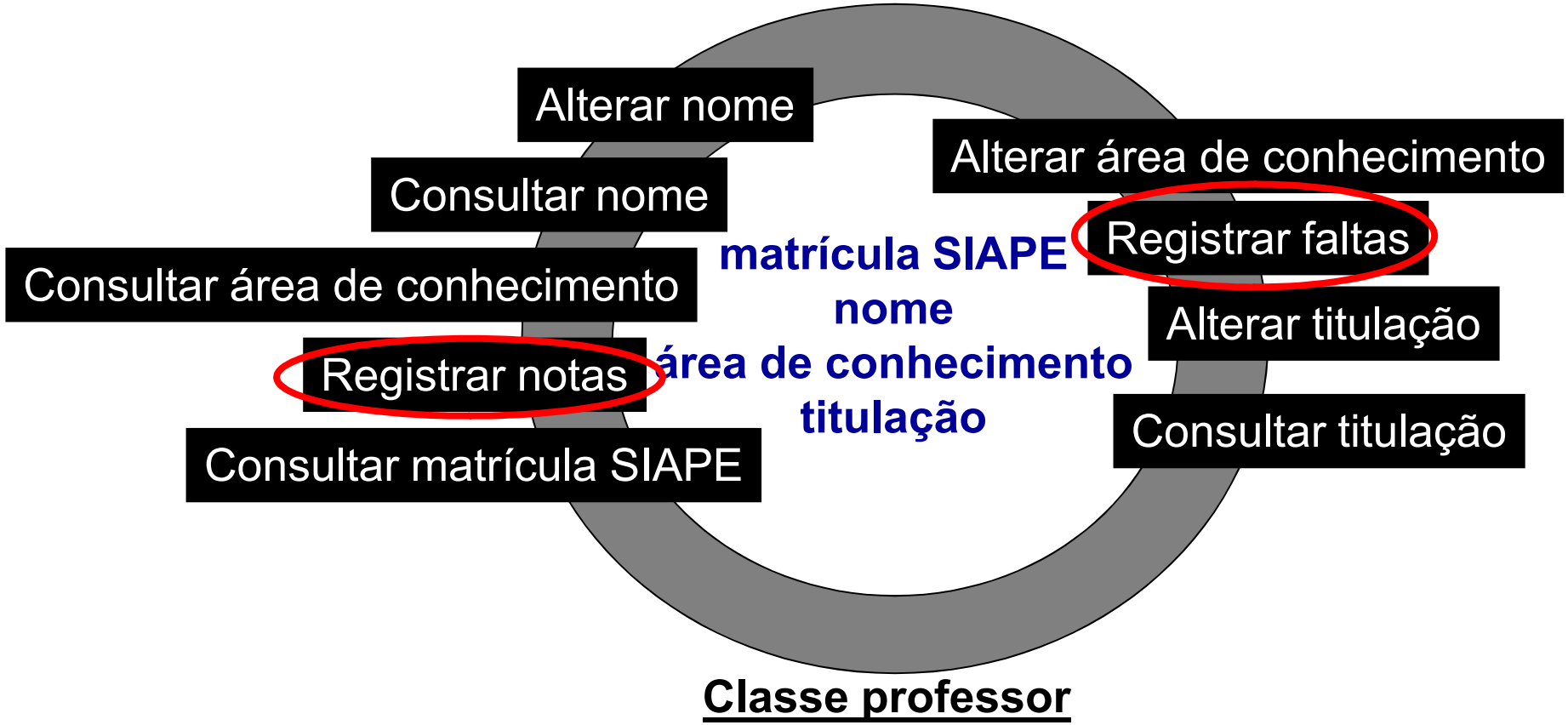
Mensagens informativas!

Conceitos/princípios da orientação a objeto



Mensagens interrogativas!

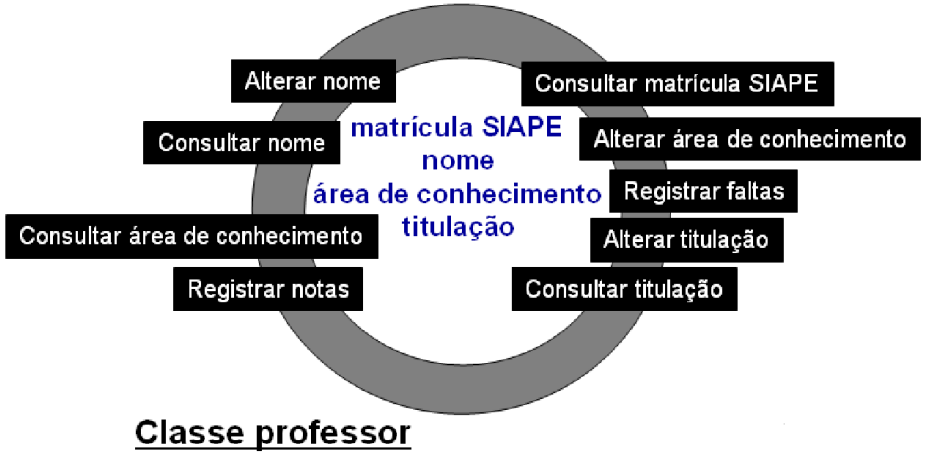
Conceitos/princípios da orientação a objeto

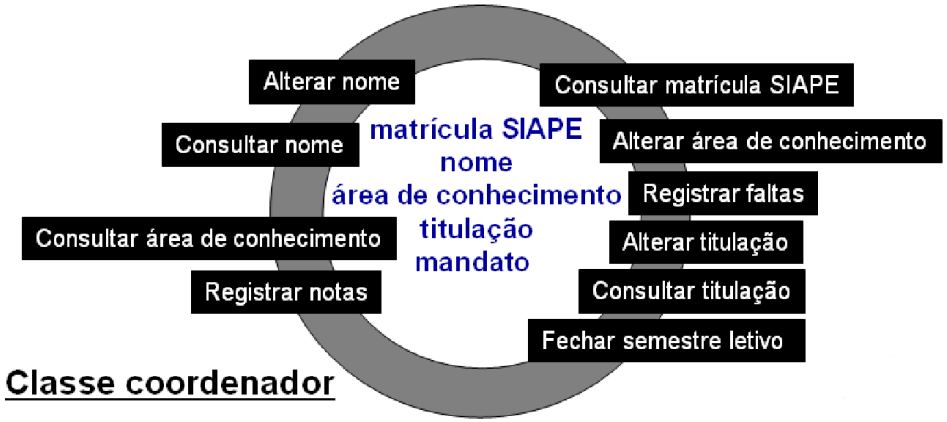


Mensagens imperativas!

Conceitos/princípios da orientação a objeto

Ao observarmos as classes:





Percebemos que?

A classe coordenador possui todos os atributos e métodos da classe professor.

Sendo assim, o que podemos fazer?

~~Replicar o código da classe professor, ajustar a cópia e renomeá-la.~~

Conceitos/princípios da orientação a objeto

Podemos idealizar um mecanismo que permita que a classe coordenador se utilize da estrutura da classe professor.

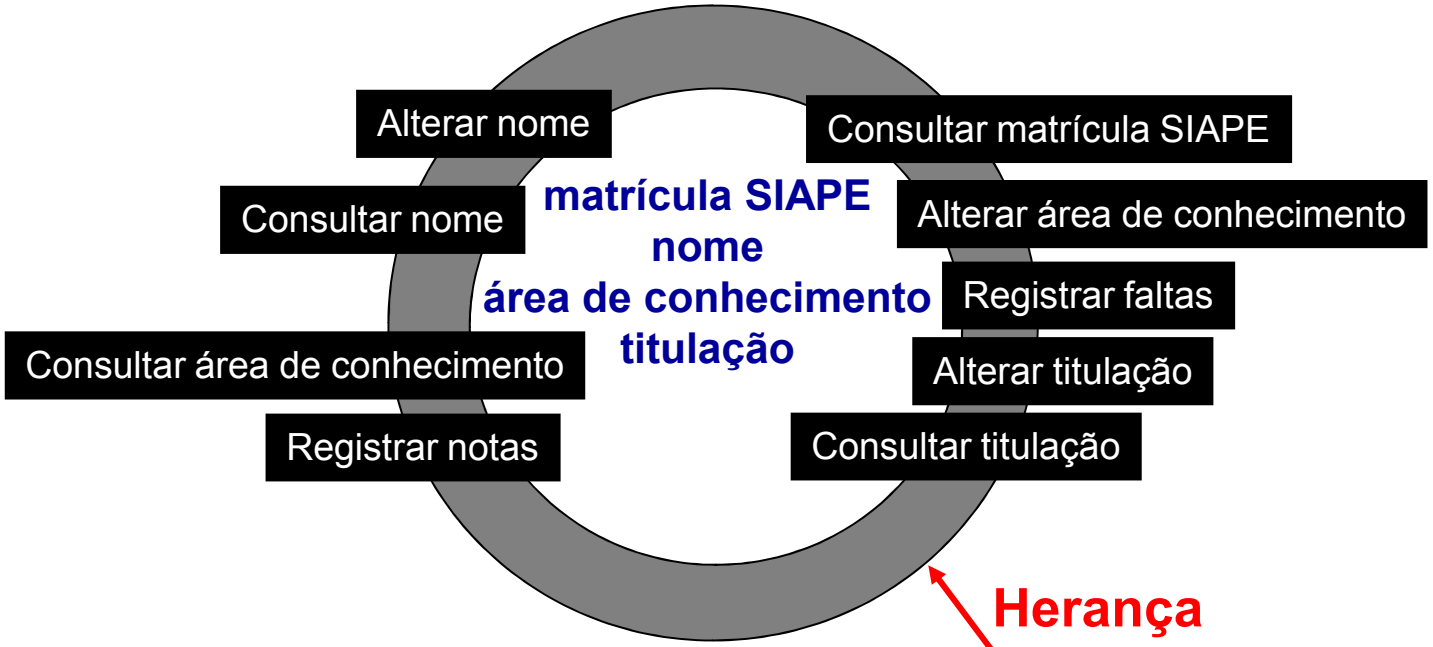
Este mecanismo existe e é denominado herança.

Em outras palavras, um tipo de dado definido como classe pode dar origem a outro tipo mediante o mecanismo de derivação por especialização chamado *herança*. Através dele, uma nova classe pode ser definida aproveitando-se o que uma classe já tem, acrescendo-se detalhes de modo a especializar a descrição (torná-la menos abstrata, mais detalhada).[8]

Neste processo a classe professor é denominada superclasse e a classe coordenador subclasse.

UNIVASF

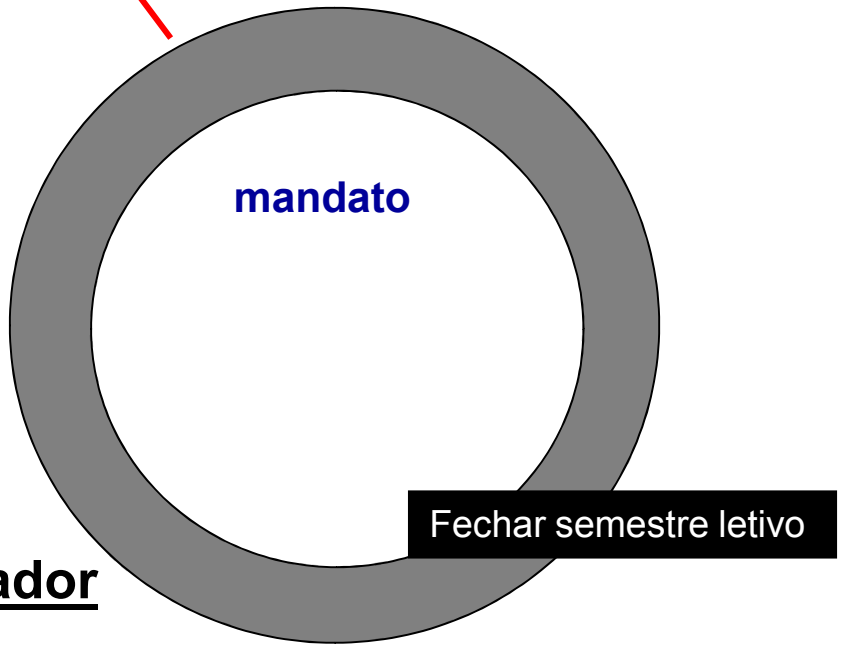
Conceitos/princípios da orientação a objeto



Herança

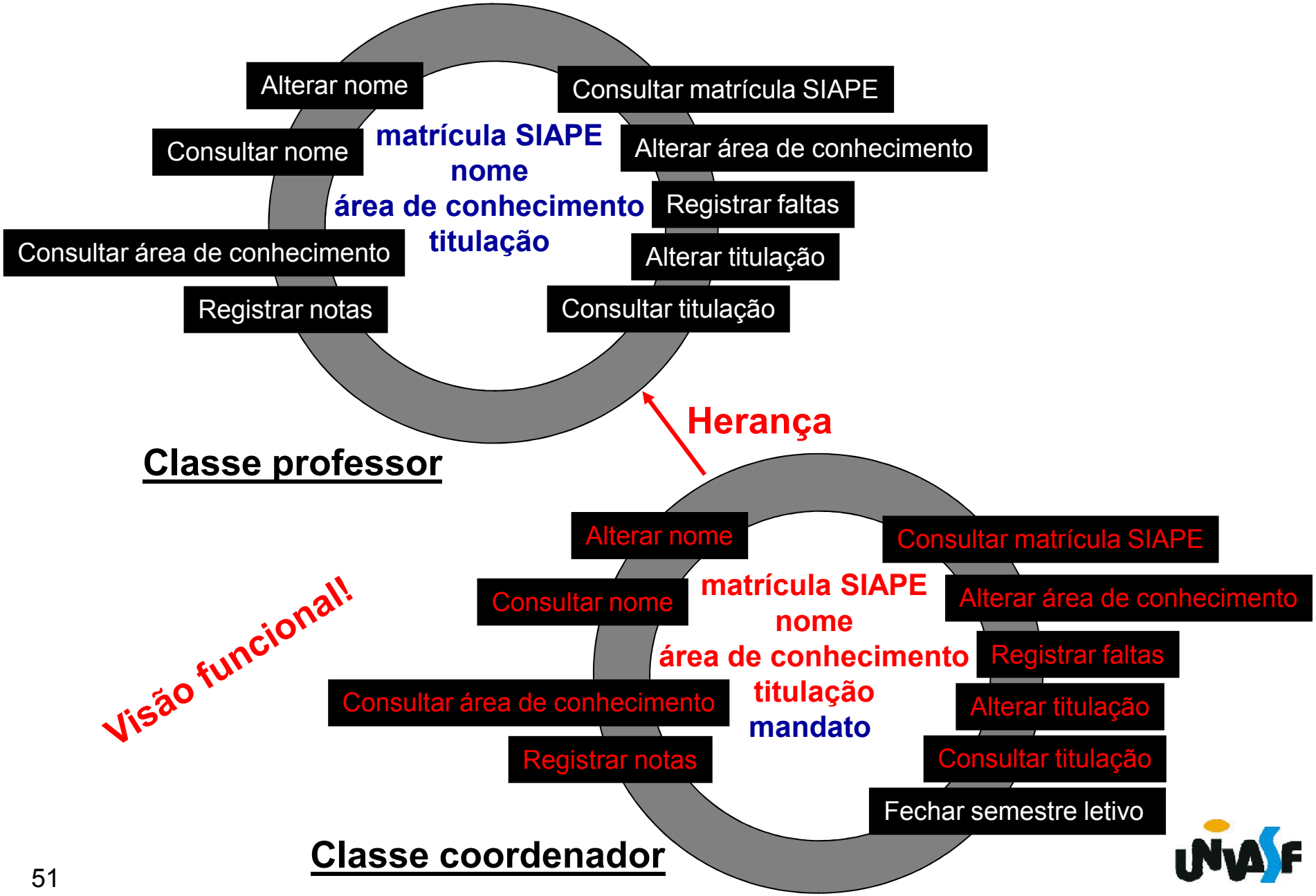
Classe professor
(Superclasse)

Visão da implementação!



Classe coordenador
(Subclasse)

Conceitos/princípios da orientação a objeto



Conceitos/princípios da orientação a objeto

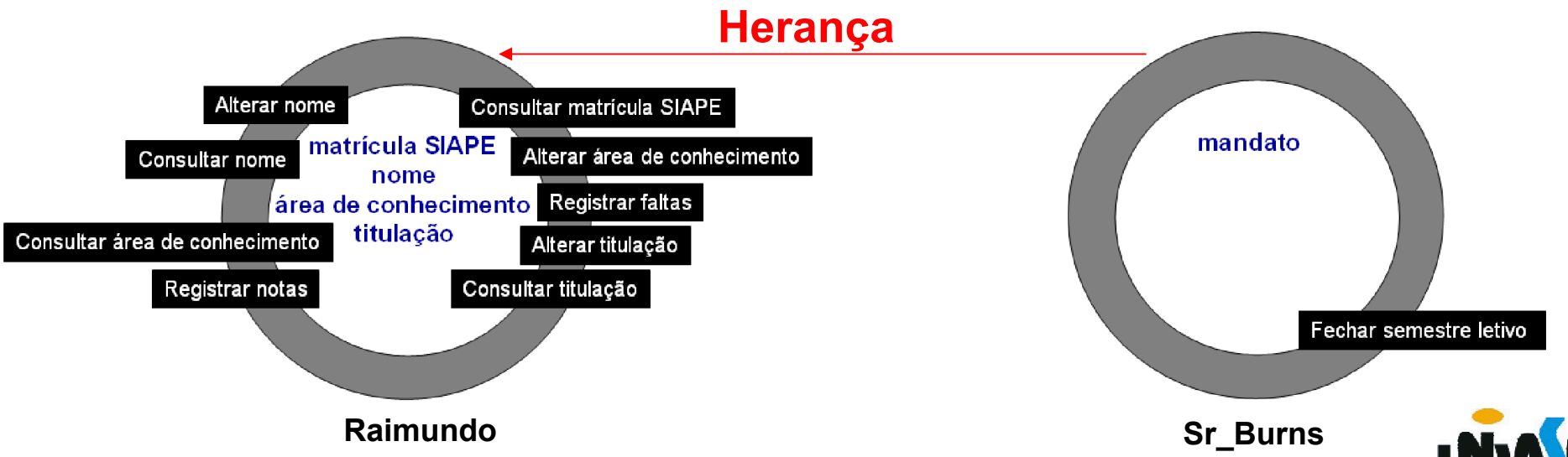
Para facilitar a visualização do mecanismo da herança analisaremos algumas mensagens:

sr_Burns.alterar_titulação(nova_titulação;)

raimundo.concultar_matrícula_SIAPE(;matricula)

sr_Burns.fechar_semestre_letivo(semestre;ok)

~~raimundo.fechar_semestre_letivo(semestre;ok)~~



Conceitos/princípios da orientação a objeto

A herança nos possibilita observar uma diferença sutil entre objeto e instância.

Você consegue visualizá-la?

Embora até agora tenhamos usado “objeto” e “instância” quase como sinônimos, vemos que a herança de uma certa forma permite que um único objeto seja simultaneamente uma instância de mais de uma classe. [8]

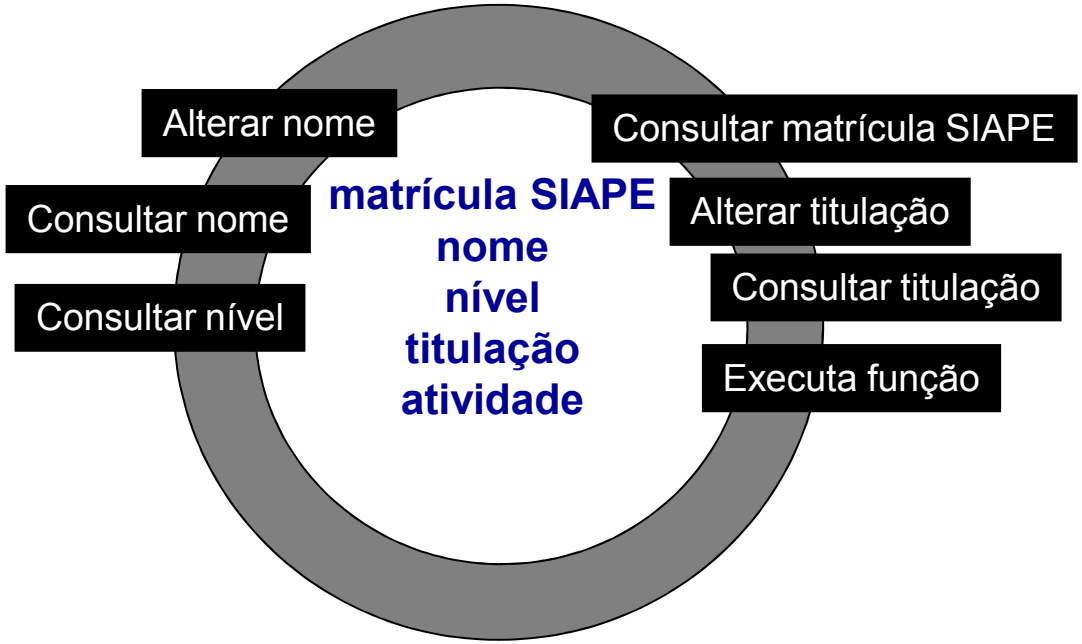
Em nosso exemplo, Sr. Burns é uma instância de coordenador e também é uma instância de professor.

UNIVASF

Conceitos/princípios da orientação a objeto

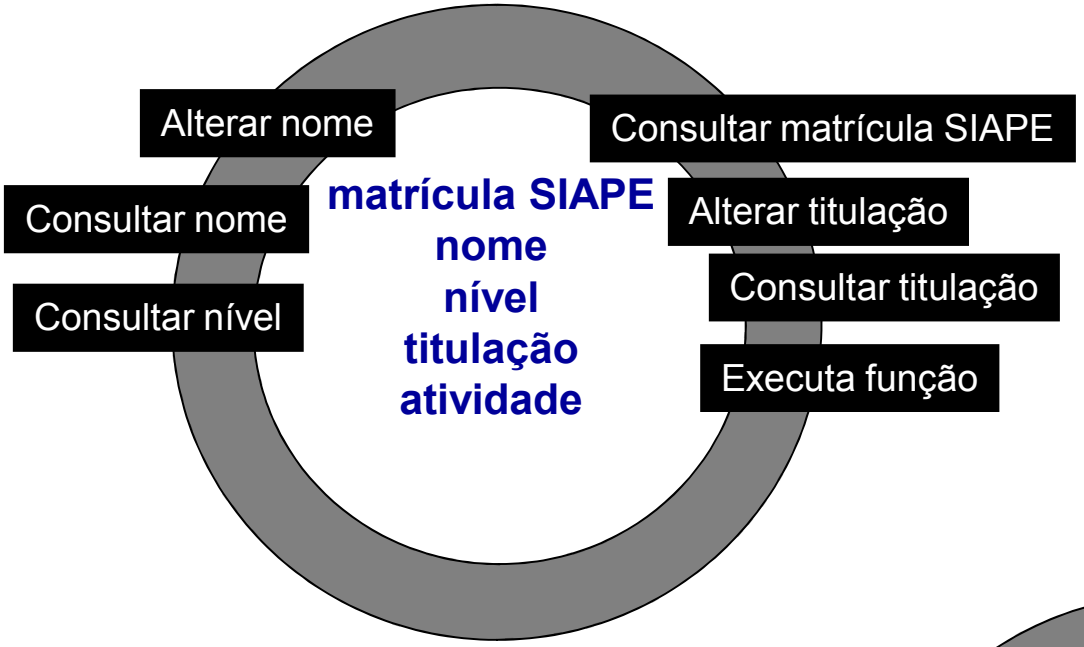
Para finalizarmos nossa análise sobre a herança, vamos imaginar a seguinte situação:

Em uma universidade temos a figura do técnico, a qual é de extrema relevância para dar suporte na realização das atividades acadêmicas.



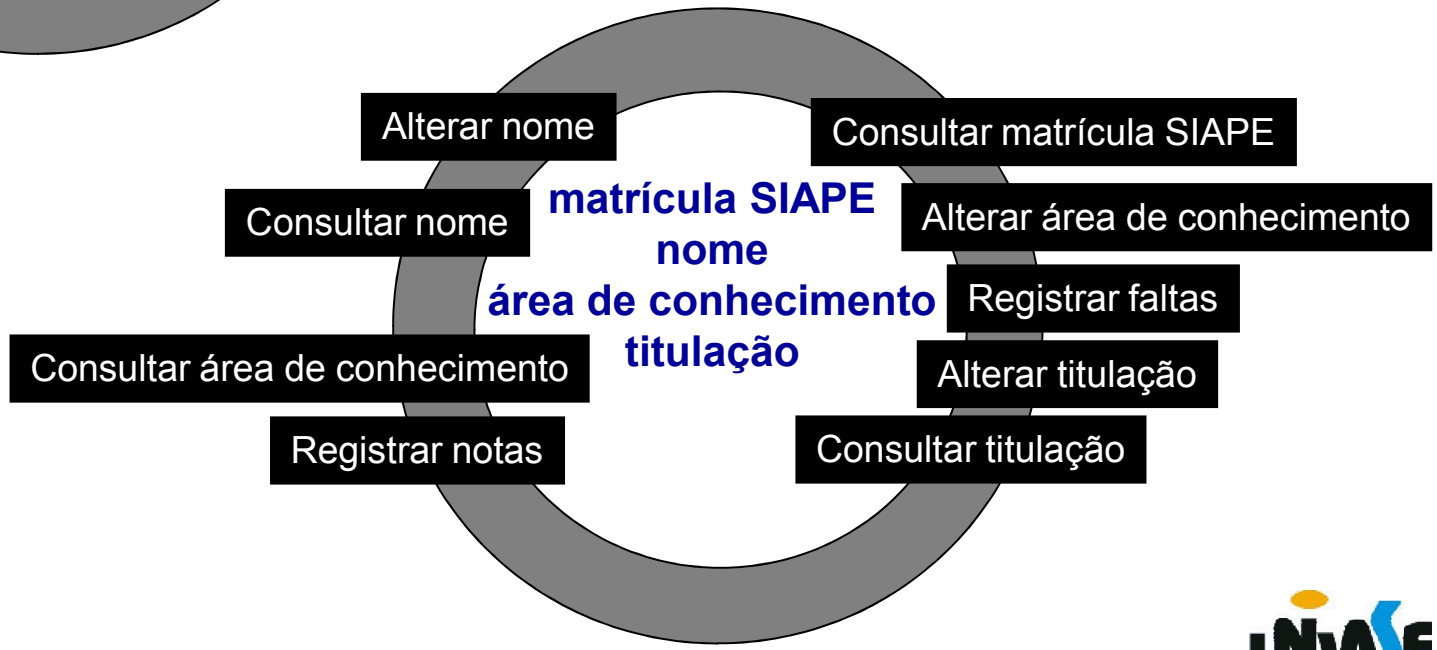
Classe técnico

Conceitos/princípios da orientação a objeto



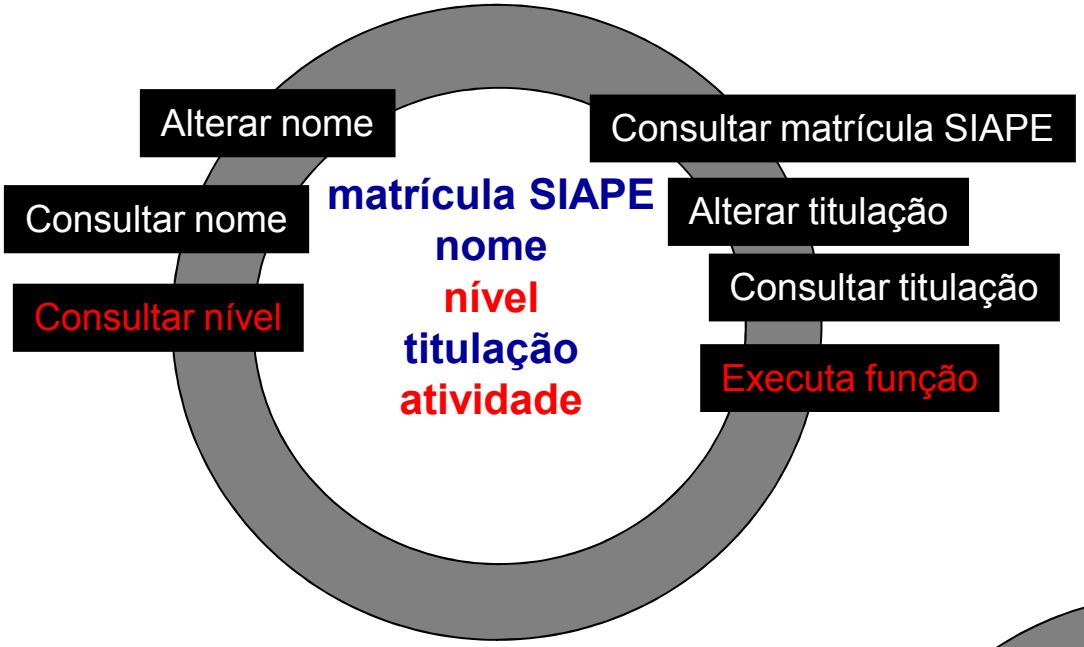
Percebemos muito em comum entre a classe técnico e a classe professor.

Classe técnico



Classe professor

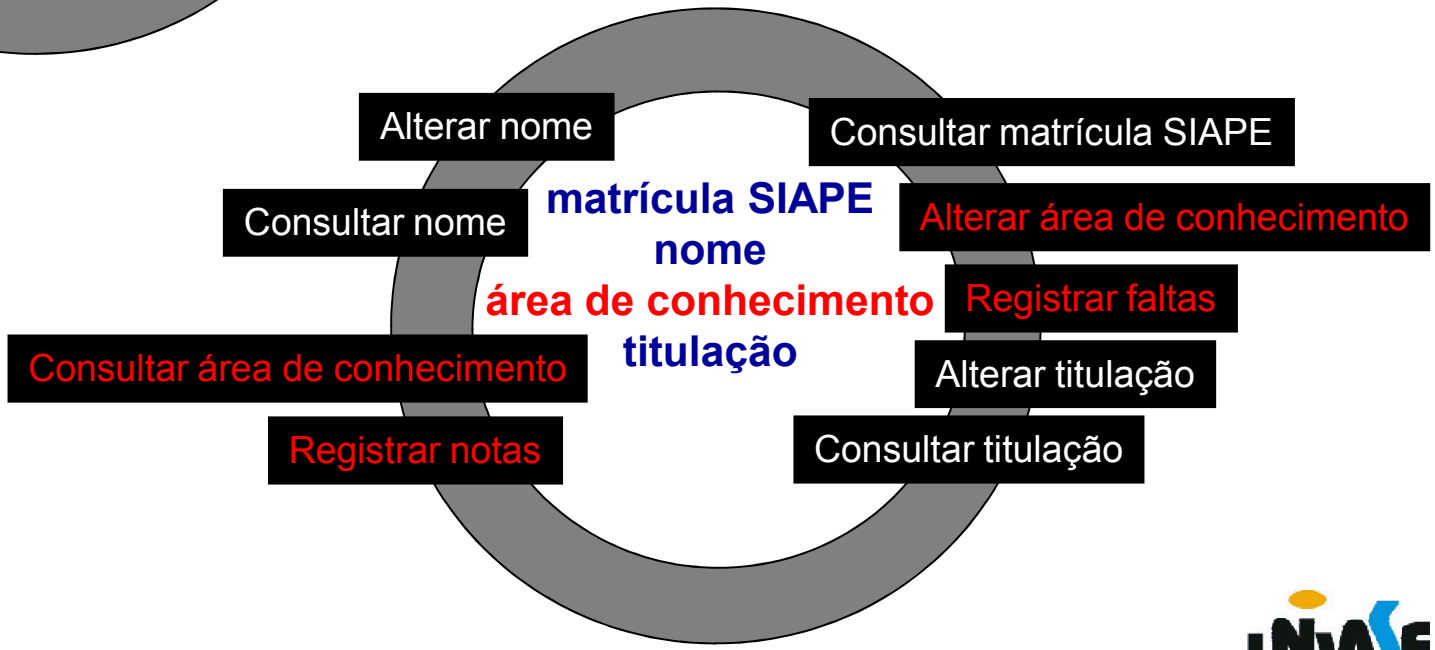
Conceitos/princípios da orientação a objeto



Classe técnico

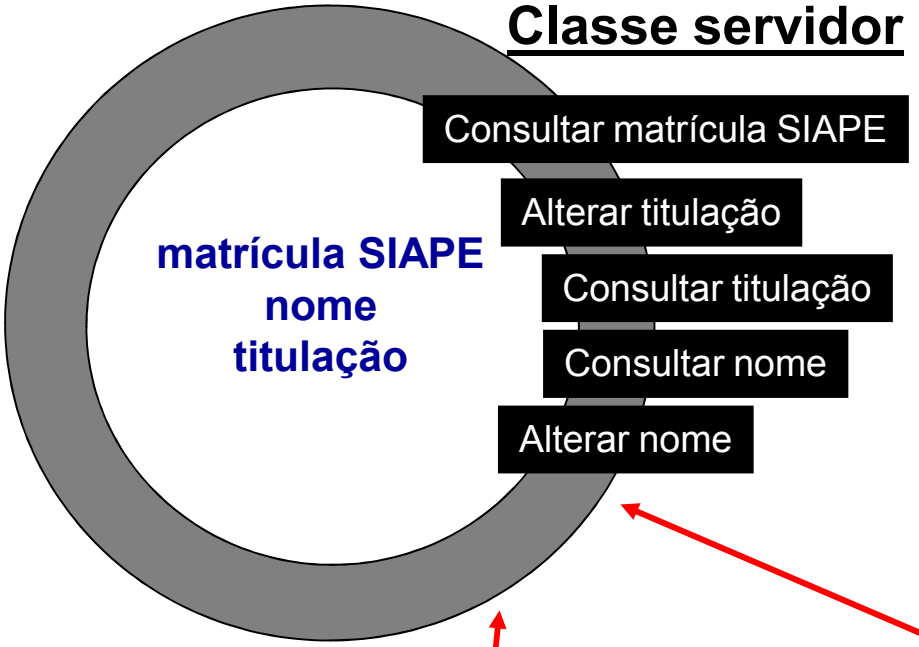
Percebemos muito em comum entre a classe técnico e a classe professor.

O que fazer?



Classe professor

Conceitos/princípios da orientação a objeto

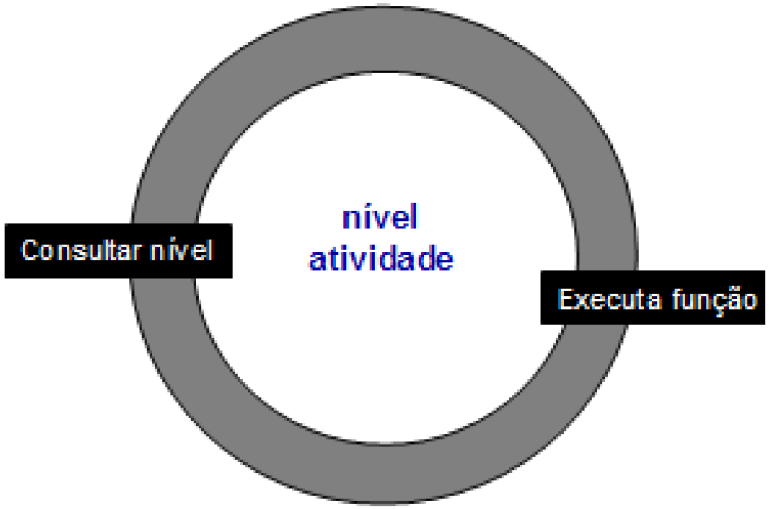


Neste caso percebemos que podemos gerar uma classe mais genérica denominada servidor, a qual será especializada na classe professor e na classe técnico.

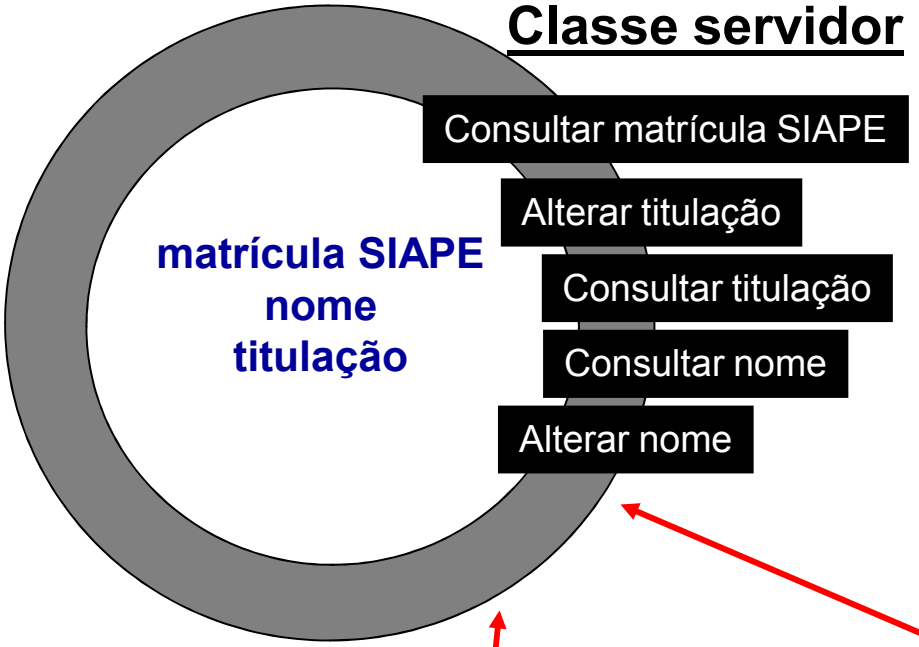
Herança

Herança

Visão da implementação!



Conceitos/princípios da orientação a objeto

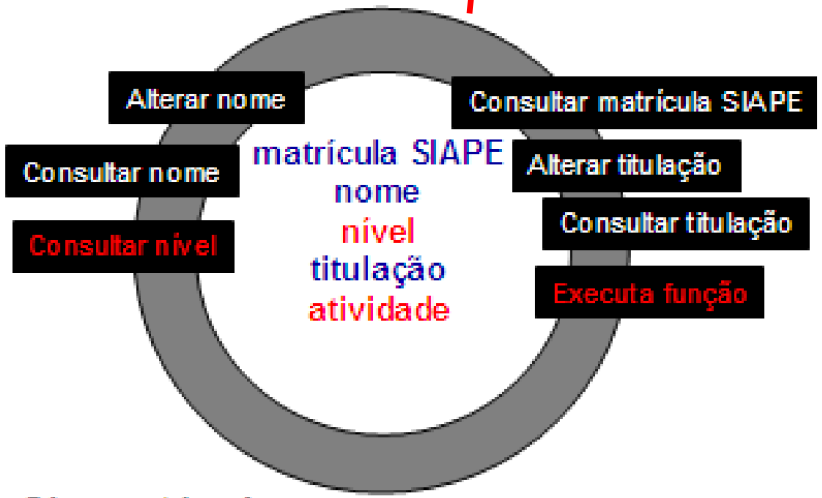


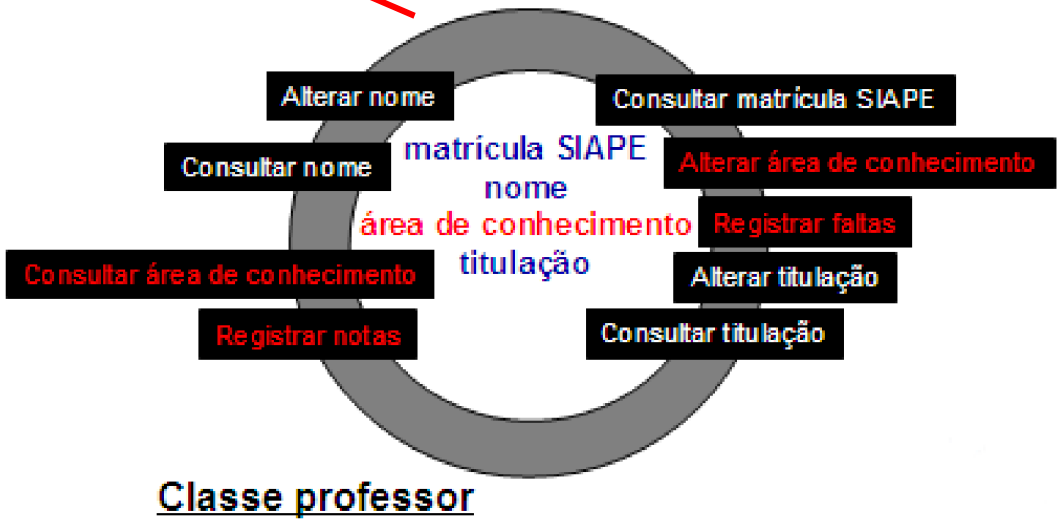
Neste caso percebemos que podemos gerar uma classe mais genérica denominada servidor, a qual será especializada na classe professor e na classe técnico.

Herança

Herança

Visão funcional!





Conceitos/princípios da orientação a objeto

Com base no que estudamos até o momento percebemos que uma subclasse possui apenas uma superclasse direta.

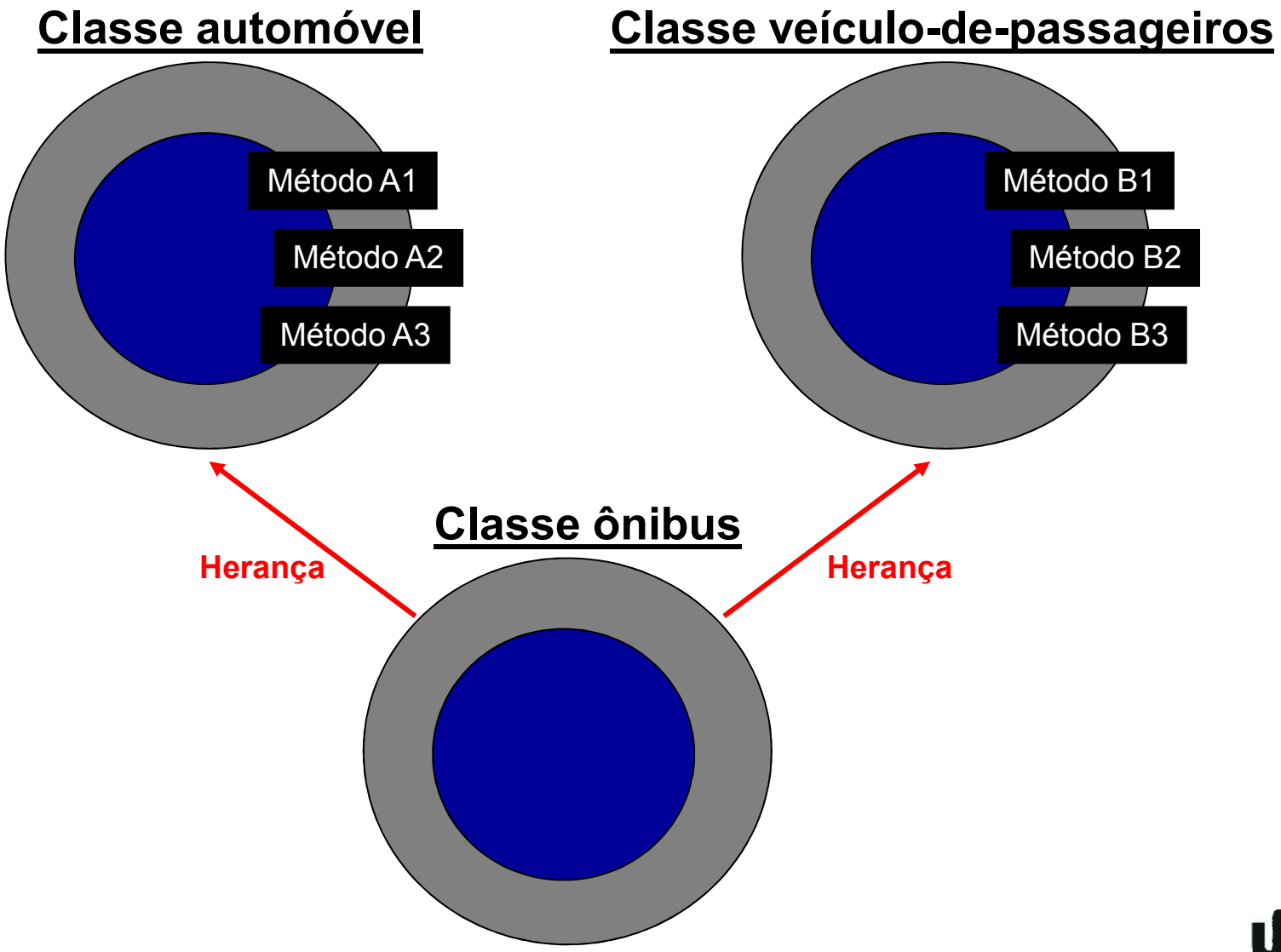
Contudo, uma análise do mundo real nos remete a perceber que em alguns sistemas temos a necessidade de uma subclasse com mais de uma superclasse.

Observe o seguinte exemplo:

- Em um determinado sistema temos uma classe automóvel;
- e uma classe veículo-de-passageiros;
- ao percebemos que existe a necessidade de criarmos a classe ônibus;
- sabemos que um ônibus é um automóvel;
- e que também é um veículo-de-passageiros.

Conceitos/princípios da orientação a objeto

Logo, teremos:



Conceitos/princípios da orientação a objeto

Um aluno atento deve ter se perguntado sobre a possibilidade de um método de uma superclasse não possuir a implementação adequada para ser aplicado em uma subclasse desta superclasse.

Neste caso o que deve ser feito?

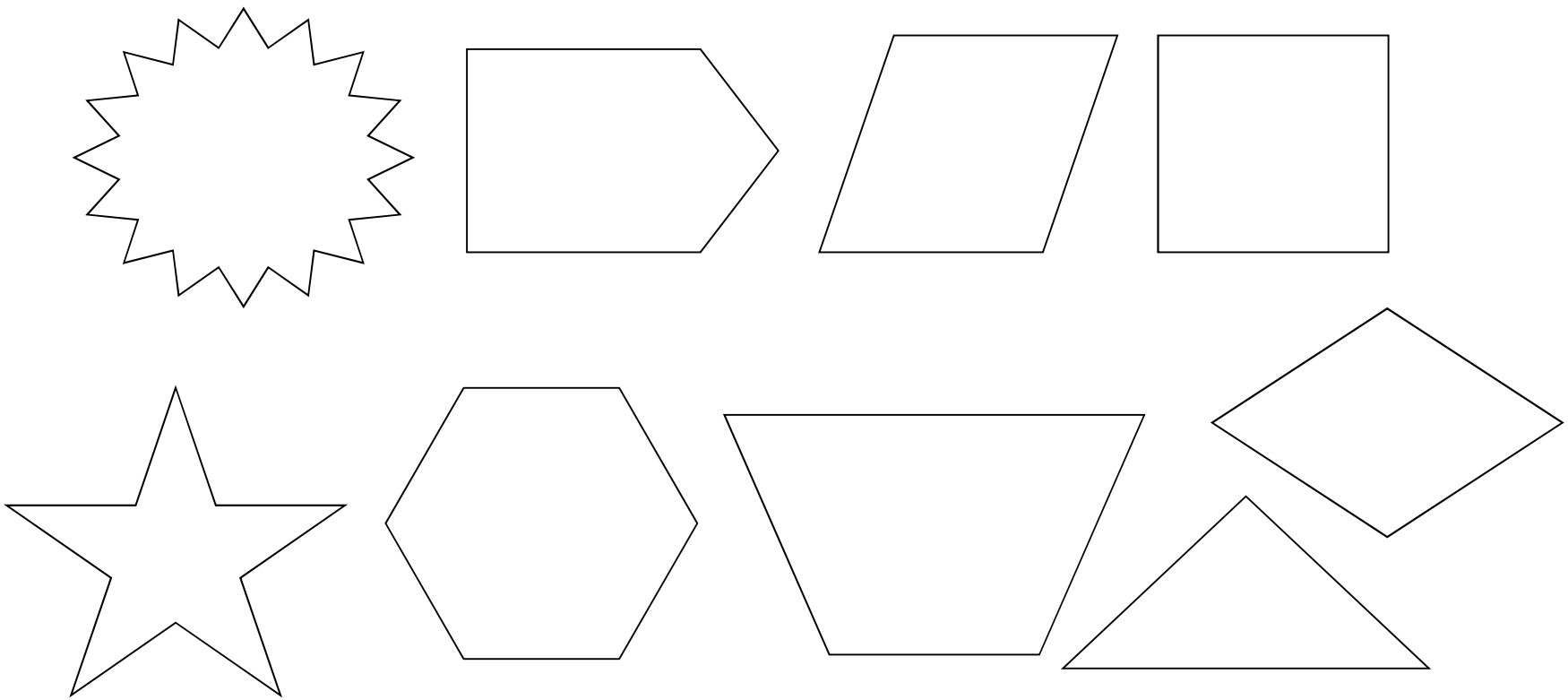
O método em questão deve ser reimplementado na subclasse.

Isso significa que passaríamos a ter um método implementado de múltiplas formas, ou seja, um método polimórfico.

UNIVASF

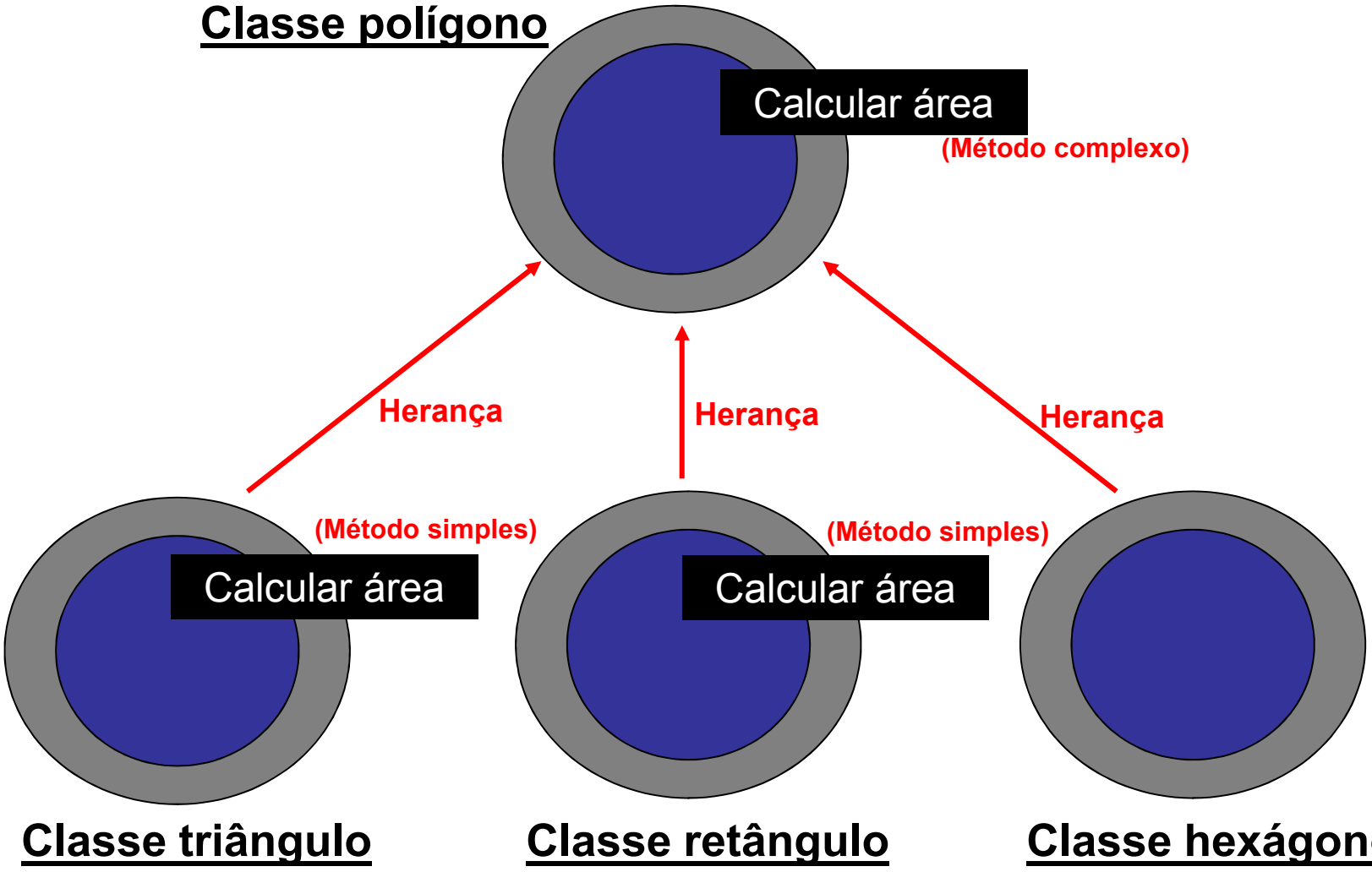
Conceitos/princípios da orientação a objeto

Para uma melhor compreensão deste princípio vamos supor a existência de uma classe polígono, que representa o tipo de forma 2-D:



Conceitos/princípios da orientação a objeto

Podemos então definir a classe polígono:



Conceitos/princípios da orientação a objeto

Vamos imaginar agora quatro objetos:

- p, instância da classe polígono;
- t, instância da classe triângulo;
- r, instância da classe retângulo;
- h, instância da classe hexágono.

Desta forma, podemos mandar a mensagem calcular área com a mesma assinatura para qualquer um dos quatro objetos acima.

Por exemplo: r.calcular_área(;area)

Pois, o método calcular área se utiliza do princípio do polimorfismo.

UNIVASF

Conceitos/princípios da orientação a objeto

Desta forma temos as seguintes definições:

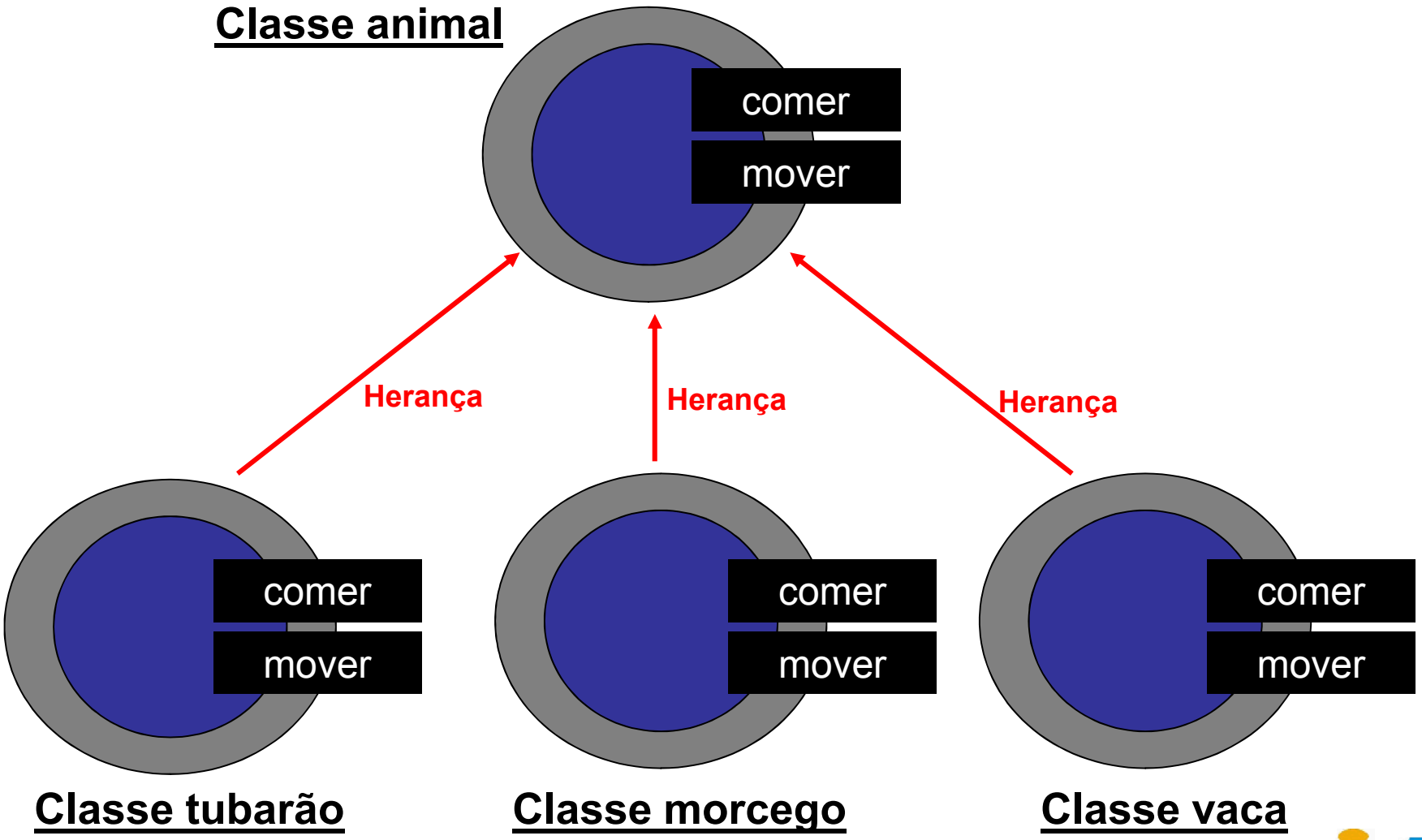
Polimorfismo é o dispositivo pelo qual o nome de um único método pode ser definido sobre mais de uma classe e pode assumir diferentes implementações em cada uma dessas classes.

Sobreposição é a redefinição de um método definido em uma superclasse em uma de suas subclasses.

UNIVASF

Conceitos/princípios da orientação a objeto

Vamos analisar agora a :



Conceitos/princípios da orientação a objeto

Observamos que a interface de todas as classes é a mesma.

Esta observação nos leva a imaginar a possibilidade de termos uma mesma variável que em instantes diferentes possua identificadores de objetos pertencentes a classes distintas.

Ou seja, em um determinado momento a variável em questão pode possuir o identificador de um objeto da classe morcego e em outro momento possuir o identificador de um objeto da classe tubarão.

Desta forma, obtém-se uma definição complementar de polimorfismo:

Polimorfismo é a propriedade pela qual uma variável pode armazenar o identificador de objetos de diferentes classes em instantes diferentes.[8]

Conceitos/princípios da orientação a objeto

O que possibilita a existência do polimorfismo?

A técnica pela qual se faz acesso à parte exata de código a ser executada somente durante o processamento (e não durante a compilação), denominada vinculação dinâmica (ou vinculação em tempo de execução (run-time)) ou vinculação tardia (late).[8]

Um conceito relacionado ao polimorfismo é a sobrecarga de um nome (ou símbolo) que ocorre quando vários métodos (ou operadores) definidos na mesma classe têm aquele nome (ou símbolo). Neste caso, se diz que o nome (ou símbolo) está sobrecarregado.

UNIVASF

Conceitos/princípios da orientação a objeto

Como de costume, um exemplo esclarecerá este tópico.

Inúmeras aplicações utiliza-se de matrizes. Logo, seria necessária uma classe Matriz. Dentre as operações que podem ser aplicadas sobre matrizes temos a multiplicação. Sendo assim, implementaríamos um método para tal.

No entanto, a multiplicação pode ocorrer entre matrizes ou entre uma matriz e um escalar.

Matriz1 * Matriz2
Matriz1 * Escalar

Observe que a operação a ser aplicada será definida com base nos operandos e não no símbolo!

Neste caso ocorreria uma sobrecarga do símbolo *. Pois, este seria associado a mais de um tipo de operação.

Conceitos/princípios da orientação a objeto

Tanto o polimorfismo quanto a sobrecarga, freqüentemente, requerem que o método específico a ser executado seja escolhido durante o processamento.

Analisaremos agora dois conceitos complementares da orientação a objeto.

Creio que vocês devem ter se perguntado se um objeto pode possuir um outro objeto como parte integrante do mesmo.

Pois, no mundo real este fato ocorre com freqüência.

Em nosso SIG@, temos um exemplo disto?

Qual?

UNIVASF

Conceitos/princípios da orientação a objeto

Em nossa análise inicial mencionamos a classe aluno e percebemos que um aluno pertence a uma turma. Ou seja, um objeto da classe turma possui objetos da classe aluno.

Desta forma, obtemos a seguinte definição:

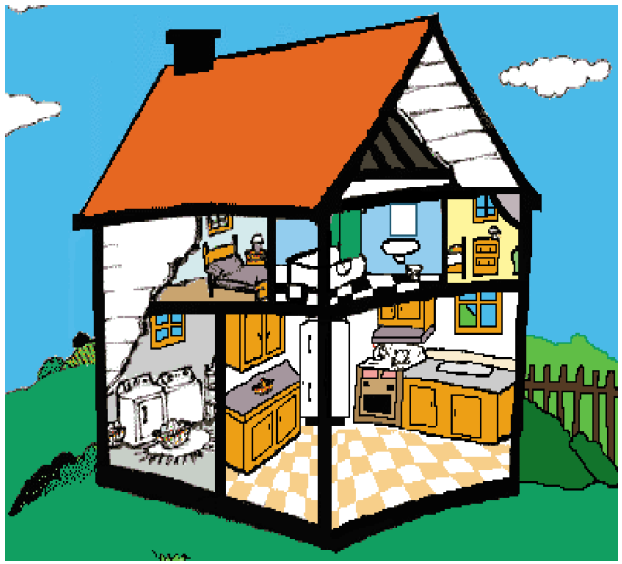
A agregação permite a construção de uma classe agregada a partir de outras classes componentes. Ou seja, podemos dizer que um objeto da classe agregada (Todo) tem objetos da classe componente (Parte).

UNIVASF

Conceitos/princípios da orientação a objeto

Fecharemos nossa análise da agregação com o seguinte exemplo:

Uma casa (todo), que é composta (partes) por portas, janelas, paredes, etc.



A pergunta a ser feita para identificar um relacionamento de agregação é: “é parte de ?”

Conceitos/princípios da orientação a objeto

Um detalhe a ser observado é que podem existir relacionamentos entre objetos das classes.

Ao descrevermos todos os relacionamentos de objetos iguais entre classes, utilizamos o conceito de associação.

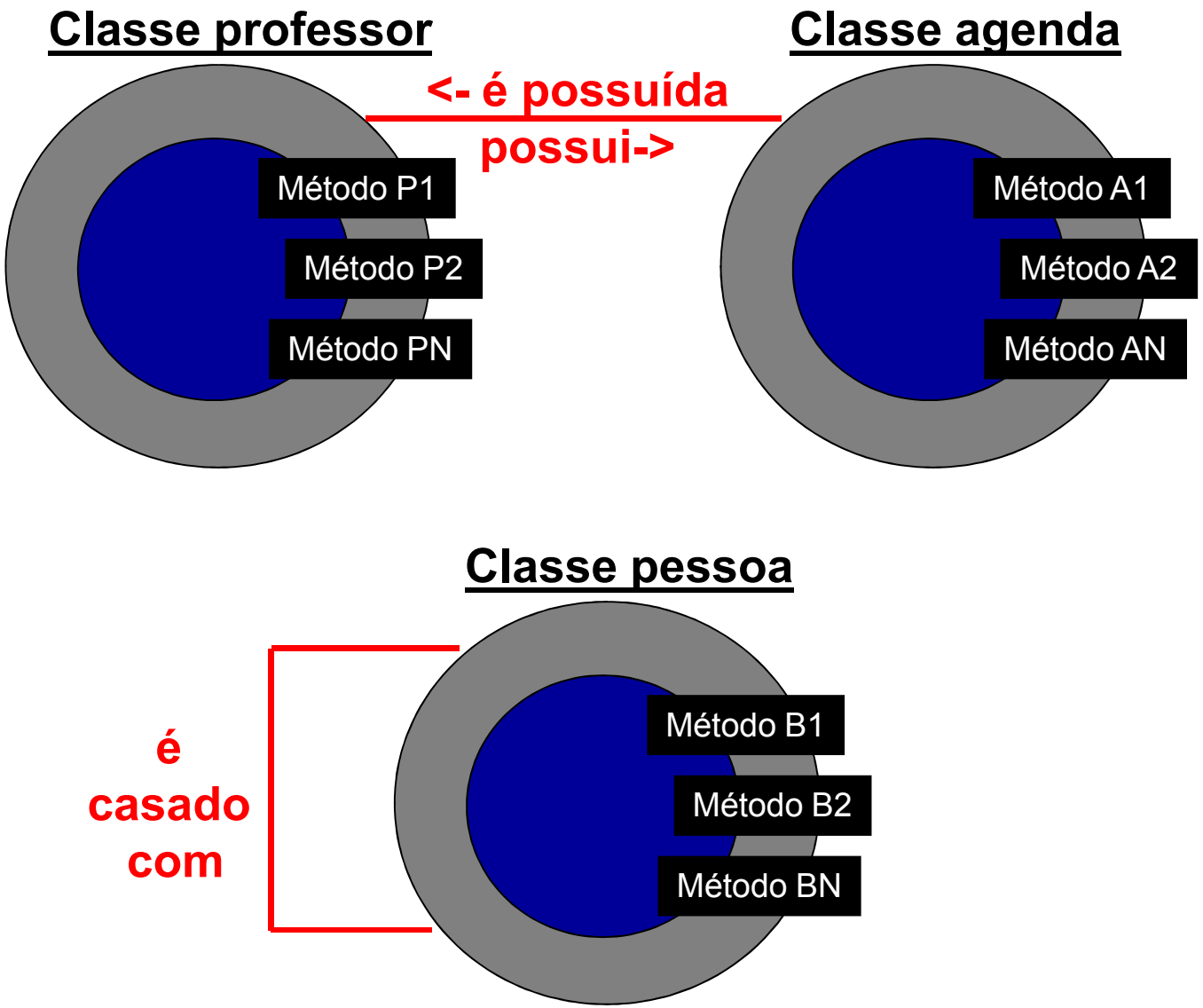
Da mesma forma que objetos são exemplares de uma classe, os relacionamentos de objetos podem ser entendidos como exemplares de uma associação.

Obs.: É possível que exista uma associação entre objetos da mesma classe, este tipo de associação é denominada reflexiva.

UNIVASF

Conceitos/princípios da orientação a objeto

Exemplos:



Introdução a UML

Introdução a UML

Neste ponto surge a questão:

Como modelar os sistemas desenvolvidos com orientação a objeto?

Utilizando um método de modelagem.

Qual método utilizar?

Na década de 1980 esta resposta era muito complexa. Pois, existiam inúmeros métodos, cada um com sua notação para representar o mesmo conceito.

UNIVASF

Introdução a UML

Esta situação gerava um crescente problema, devido ao grande avanço na utilização da orientação a objeto (OO) no desenvolvimento de sistemas.

Percebendo isto, em 1994 dois conceituados metodologistas (James Rumbaugh e Grady Booch) efetuaram a unificação de seus métodos. Buscando minimizar os transtornos gerados pela ausência de uma metodologia padrão para OO.

O método foi publicado sob o título Unified Method 0.8.

No ano de 1995 outro metodologista, Ivar Jacobson, se juntou à dupla, gerando em outubro de 1996 a versão 0.91 da Unified Modeling Language ou apenas UML.

UNIVASF

Introdução a UML

Desde então a UML vem se consolidando cada vez mais como linguagem padrão para modelagem de sistemas orientados a objeto.

Numa definição mais geral UML é uma linguagem para especificação, construção, visualização e documentação de sistemas de software. Representando a união da sintaxe gráfica de vários métodos, com vários símbolos removidos e vários adicionados.

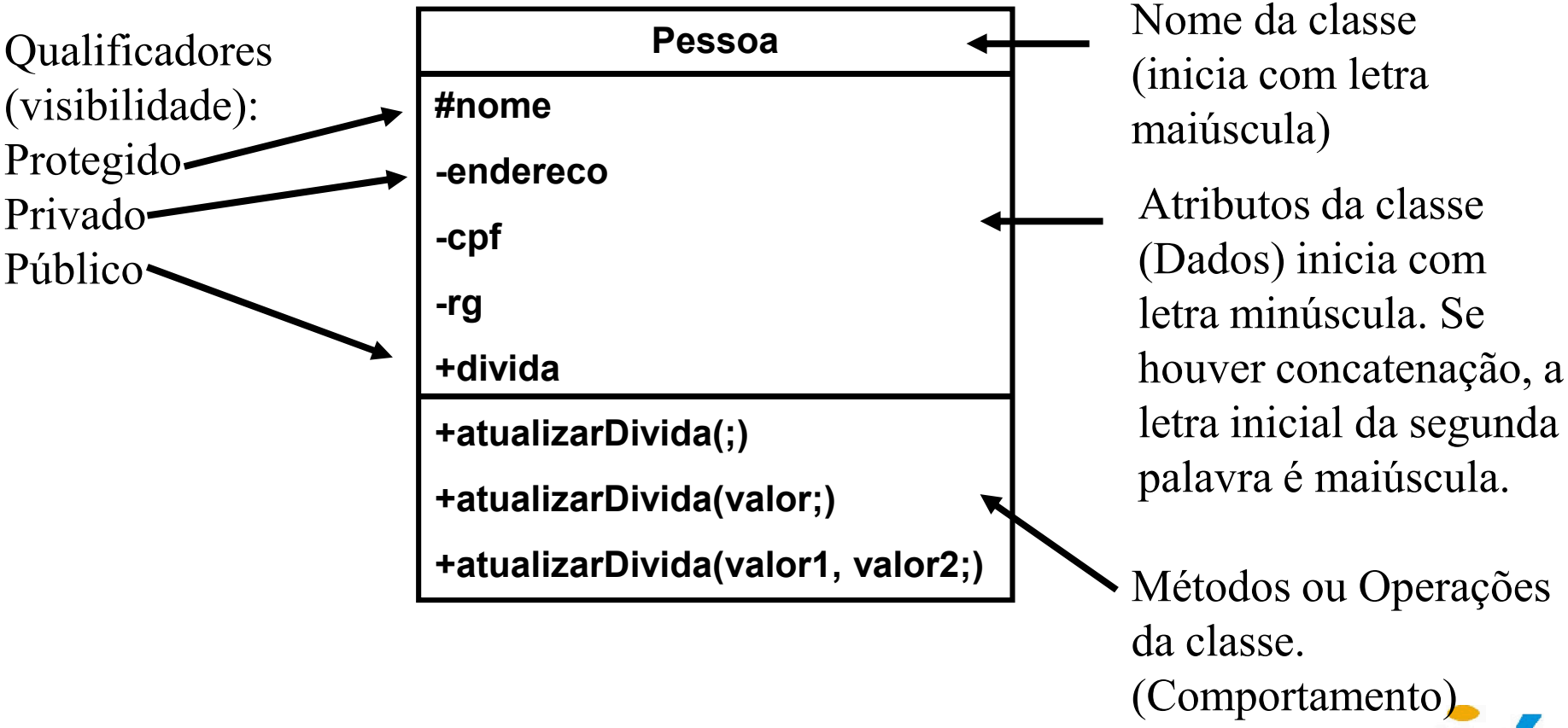
Detalharemos agora o **diagrama de classes em UML**.

UNIVASF

Introdução a UML

Uma classe em UML é representada por um retângulo dividido em três partes.

Por exemplo:



Introdução a UML

Existem diversos softwares para gerar diagramas em UML.

Trabalharemos com o **Dia**, pois este serve para criar inúmeros tipos de diagramas, como por exemplo, diagramas UML, diagramas de fluxos, diagramas de rede e diagramas elétricos.

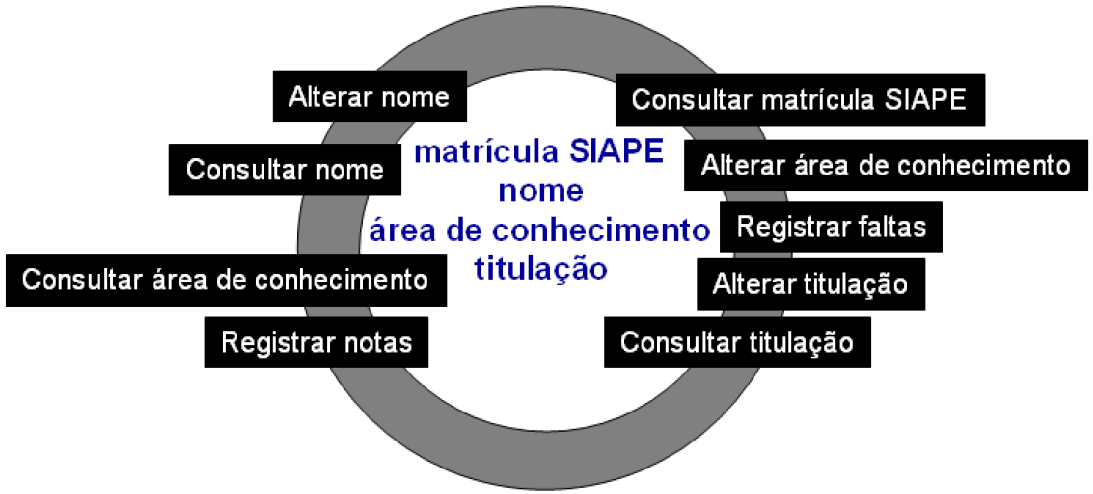
Duas característica do Dia influenciaram em sua escolha, o fato deste ter versões para Linux e para Windows além de ser um software gratuito.

UNIVASF

Introdução a UML

Exercício: Represente a classe professor, discutida anteriormente, em UML.

Professor
-matriculaSIAPE -nome -areaDeConhecimento -titulacao
+alteraNome (nome;) +consultaNome(;nome) +consultaAreaDeConhec (;area) +registraNotas(turma;) +consultaMatSIAPE(;mat) +alterarAreaDeConhec(area;) +registraFaltas(turma;) +alteraTitulacao(titulacao;) +consultaTitulacao(;titulacao)

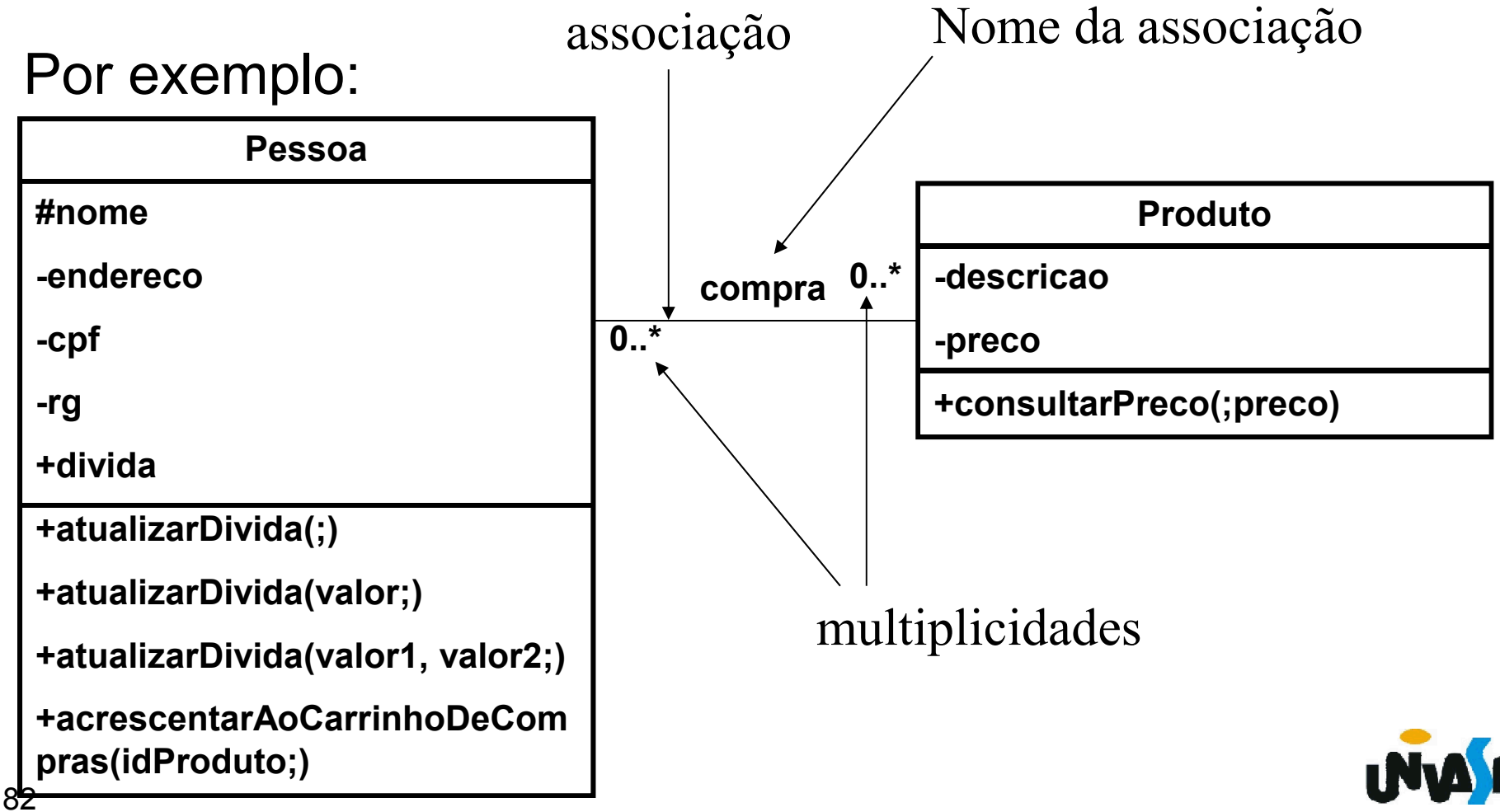


Classe professor

Introdução a UML

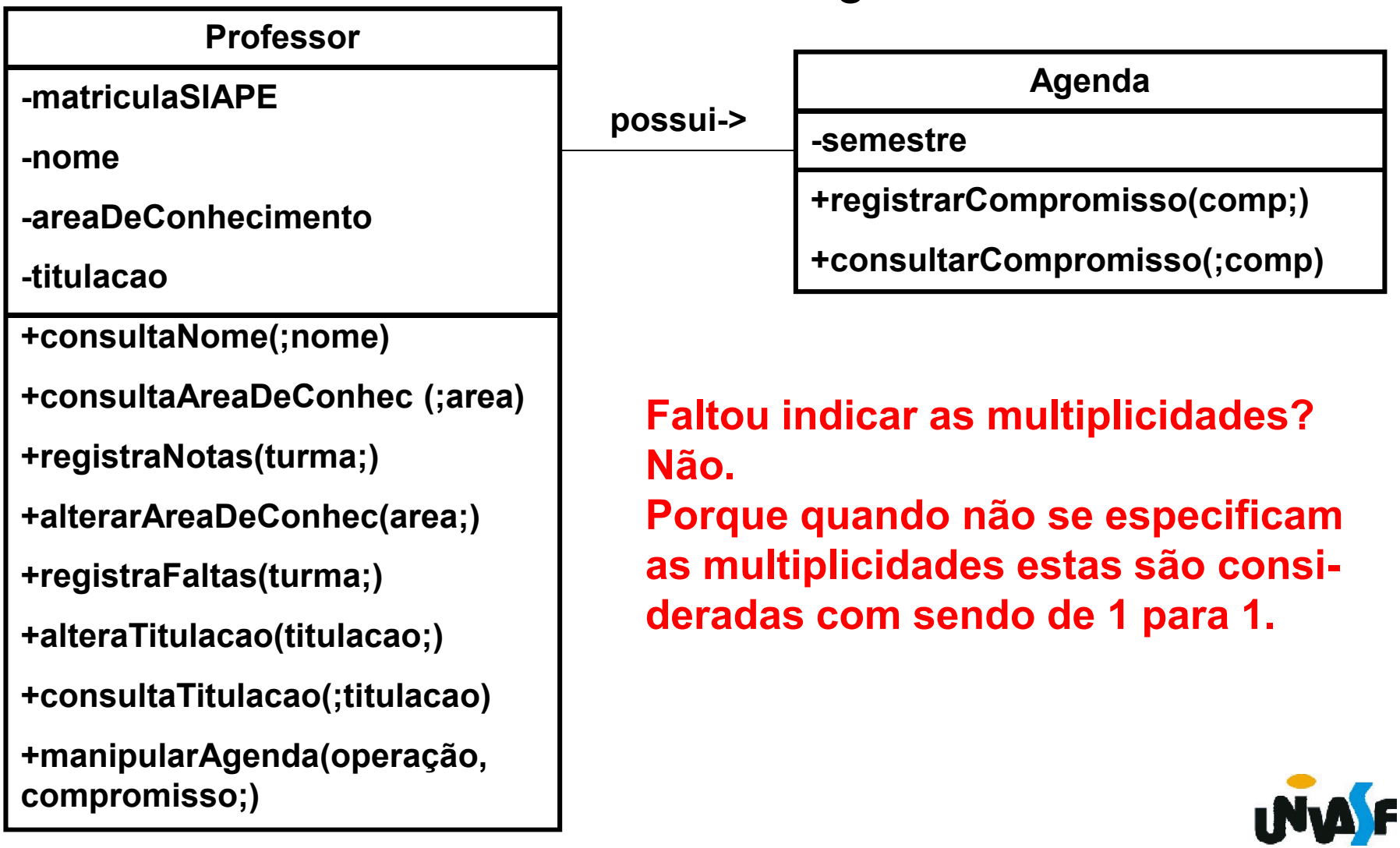
Uma associação em UML é representada por uma linha ligando os retângulos que representam as classes envolvidas.

Por exemplo:



Introdução a UML

Exercício: Represente uma associação, utilizando UML, considerando um sistema de gestão acadêmica.



Faltou indicar as multiplicidades?
Não.
Porque quando não se especificam as multiplicidades estas são consideradas com sendo de 1 para 1.