

Elementos de um programa orientado a objetos

Prof. Murillo G. Carneiro
FACOM/UFU

Material baseado nos slides disponibilizados pelo Prof. Ricardo Pereira e Silva (UFSC)

Objetivo

- Mostrar os elementos que compõem um programa orientado a objetos
 - Em tempo de desenvolvimento
 - Em tempo de execução
- Destacar que a modelagem orientada a objetos tem a obrigação de descrever um software tanto em tempo de desenvolvimento quanto em tempo de execução

O paradigma de orientação a objetos

- Abordagem em que qualquer realidade é descrita como um conjunto de classes
 - Cada classe possui um conjunto de atributos e métodos
- Linguagens de programação orientadas a objetos refletem essa estrutura: conjunto de classes

O paradigma de orientação a objetos

Enxergar um software orientado a objetos como um conjunto de classes é apenas uma visão parcial

Tempo de desenvolvimento X tempo de execução

- Em tempo de desenvolvimento o foco é o conjunto de classes que compõe o software
- Em tempo de execução as classes originam instâncias que interagem

Tempo de desenvolvimento

- Um programa orientado a objetos escrito em uma linguagem de programação como C++, Java ou Smalltalk é um conjunto de classes
 - Noção básica de quem passa por um curso de programação orientada a objetos
 - Ambientes de suporte à produção de código dão essa visão
- Produzir um programa → desafio de produzir o conjunto de classes que satisfaça aos requisitos estabelecidos

Identificar classes

- Classes devem modelar os elementos do domínio do problema tratado
- Por exemplo, para um programa de Jogo-da-velha são esperadas classes como Tabuleiro, Jogador, Posicao

Classes de um programa orientado a objetos – exemplo

Tabuleiro

Jogador

Posicao

Preencher as classes

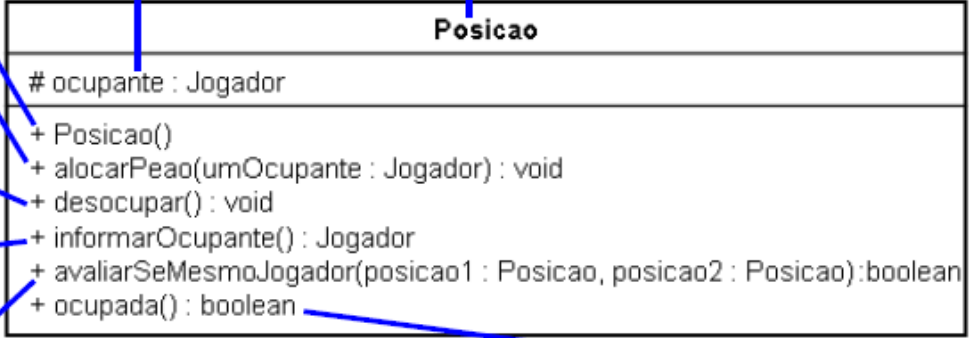
- Classes → conjunto de caixas vazias
- Conteúdo das classes
 - Conteúdo de qualquer programa, independente de paradigma → dados e instruções que manipulam esses dados
 - Atributos → dados alocados em uma classe
 - Métodos → procedimentos alocados em uma classe
- Desafio → definir o conjunto de atributos e métodos de cada classe

Classe “preenchida” – exemplo

Posicao
ocupante : Jogador
+ Posicao() + alocarPeao(umOcupante : Jogador) : void + desocupar() : void + informarOcupante() : Jogador + avaliarSeMesmoJogador(posicao1 : Posicao, posicao2 : Posicao):boolean + ocupada() : boolean

O código orientado a objetos – exemplo

```
public class Posicao {  
    protected Jogador ocupante;  
    public Posicao() {  
    }  
    public void alocarPeao(Jogador umOcupante) {  
        this.ocupante = umOcupante;  
    }  
    public void desocupar() {  
        this.ocupante = null;  
    }  
    public Jogador informarOcupante() {  
        return this.ocupante;  
    }  
    public boolean avaliarSeMesmoJogador(Posicao posicao1, Posicao posicao2) {  
        boolean confirmado = false;  
        if ( (this.ocupada() ) && (posicao1.ocupada() && posicao2.ocupada()) )  
            confirmado = ( posicao1.informarOcupante() == posicao2.informarOcupante() )  
                && (this.informarOcupante() == posicao2.informarOcupante() );  
        if (confirmado)  
            this.informarOcupante().assumirVencedor(true);  
        return (confirmado)  
    }  
    public boolean ocupada() {  
        return (this.ocupante != null);  
    }  
}
```



The diagram shows a class named **Posicao**. It has a private attribute **# ocupante : Jogador**. The methods listed are: **+ Posicao()**, **+ alocarPeao(umOcupante : Jogador) : void**, **+ desocupar() : void**, **+ informarOcupante() : Jogador**, **+ avaliarSeMesmoJogador(posicao1 : Posicao, posicao2 : Posicao):boolean**, and **+ ocupada() : boolean**. Blue lines connect the code in the Java snippet to the corresponding elements in the UML diagram: the class name, the protected attribute, the constructor, the `alocarPeao` method, the `desocupar` method, the `informarOcupante` method, the `avaliarSeMesmoJogador` method, and the `ocupada` method.

O desafio de identificar os elementos de um programa

- O dilema em relação à composição das classes é a identificação e distribuição de atributos e métodos
 - Por que *Posicao* referencia o seu ocupante e não o oposto?
 - Por que a responsabilidade de alocação de peão (método *alocarPeao*) foi dada à classe *Posicao* e não a alguma outra?
- Não é uma questão simples...

A dificuldade de manusear apenas a visão de tempo de desenvolvimento

- Ambientes de apoio à codificação dão APENAS a visão do tempo de desenvolvimento
 - As classes com seus atributos e métodos
 - Eclipse, Visual C++, VisualWorks etc.
- Suponha um software com 200 classes, com média de 20 métodos por classe → como identificá-los?

A dificuldade de manusear apenas a visão de tempo de desenvolvimento

EM SUMA: tentar definir o código em tempo de desenvolvimento, sem a visão dos elementos do tempo de execução, é como tentar dirigir um carro na escuridão, com faróis apagados

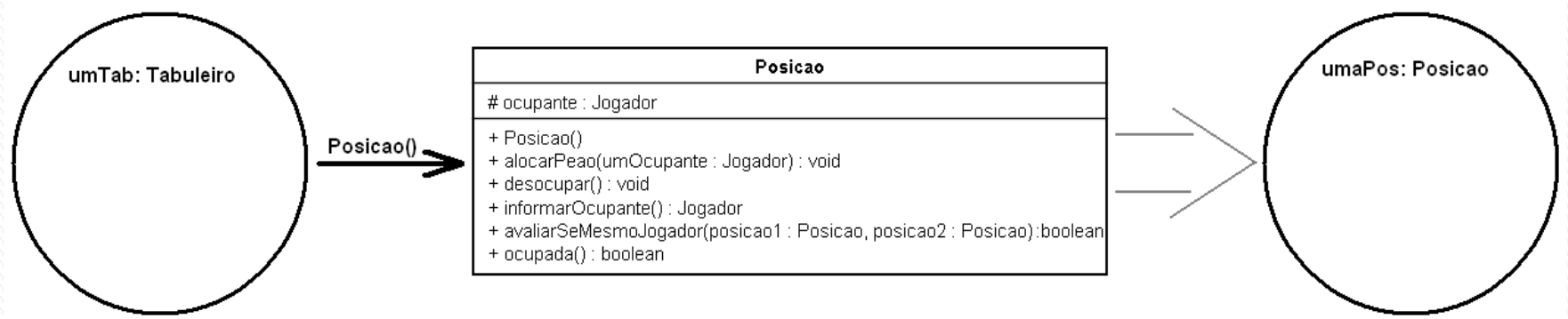
Tempo de execução

- Elementos centrais → objetos
 - Instâncias das classes
- Mais que ver um conjunto de objetos: observar a interação entre os objetos
 - Ocorre ao longo do tempo
 - Envios de mensagens
 - Método do destinatário invocado pelo emissor
- Tempo de execução, similar a um filme → tempo de desenvolvimento, a uma foto

Um exemplo: Jogo-da-velha

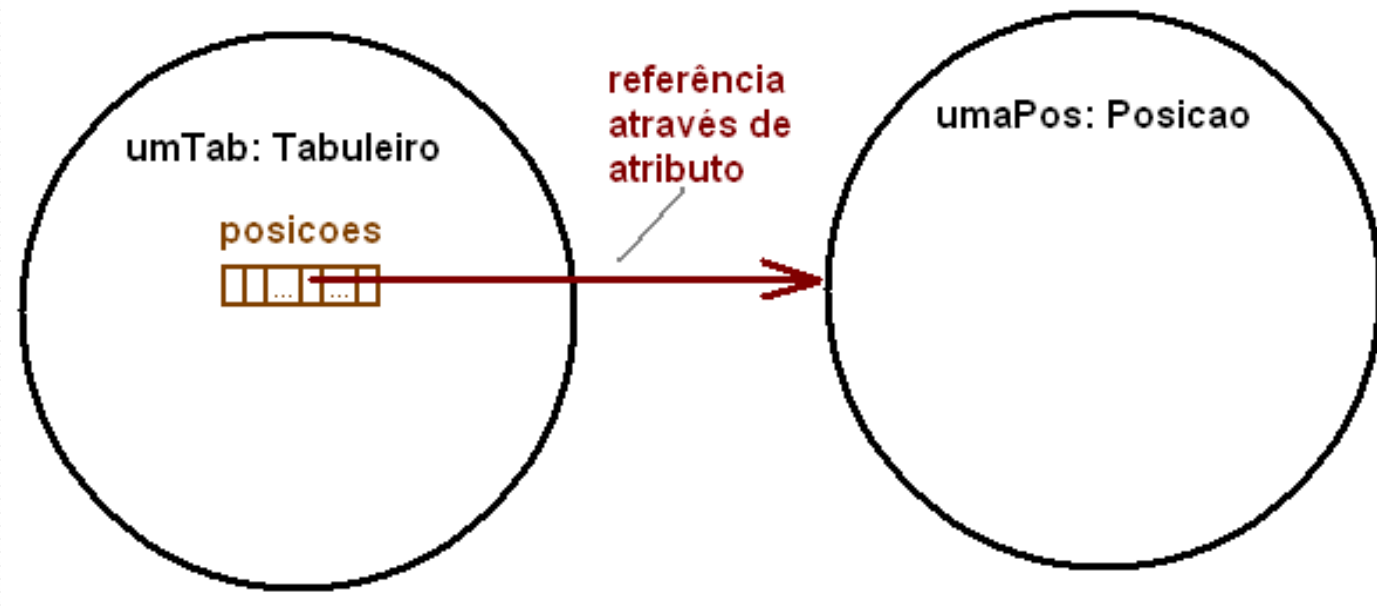
- Supor três situações que ocorrem ao longo da execução do programa
 1. Criação de uma instância da classe *Posicao*
 2. Execução do método *alocarPeao*, da classe *Posicao*
 3. Ação do tabuleiro em resposta à seleção de uma posição
- Como vê-las ocorrendo?
- A seguir uma modelagem gráfica dessas situações, sem usar UML

1 – Criação de uma instância



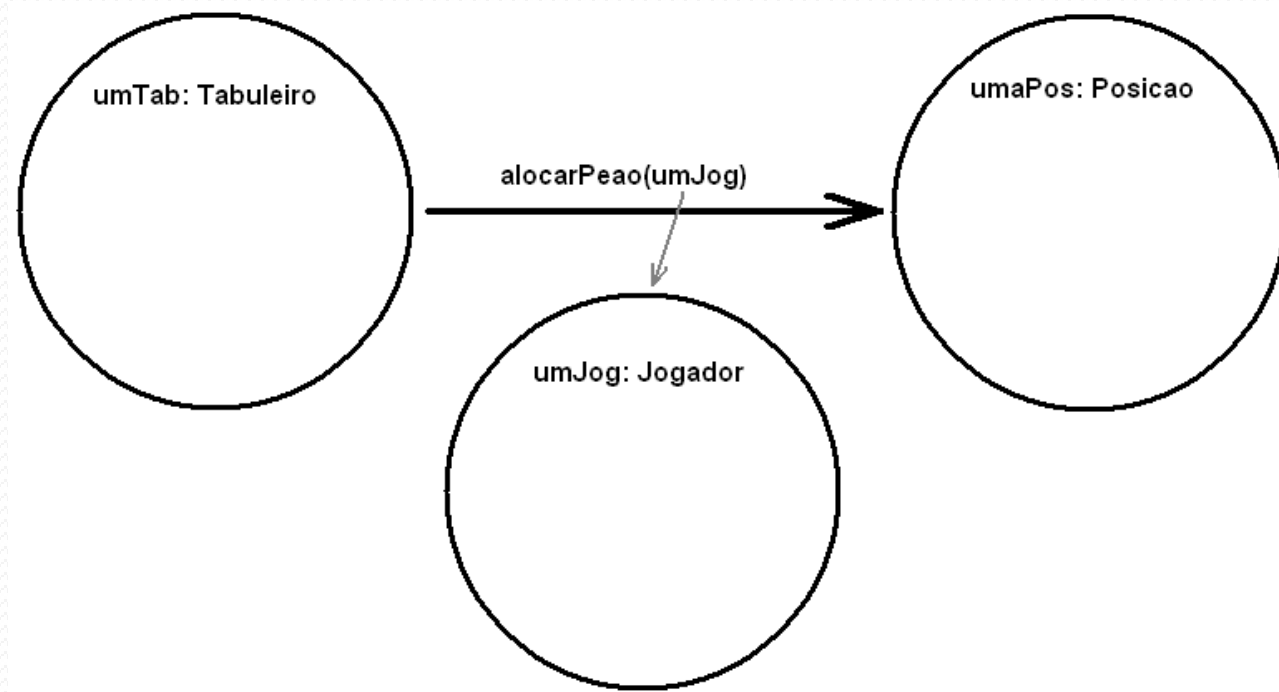
- Instância da classe *Tabuleiro* (*umTab*) invoca o método construtor da classe *Posicao*

1 – Criação de uma instância



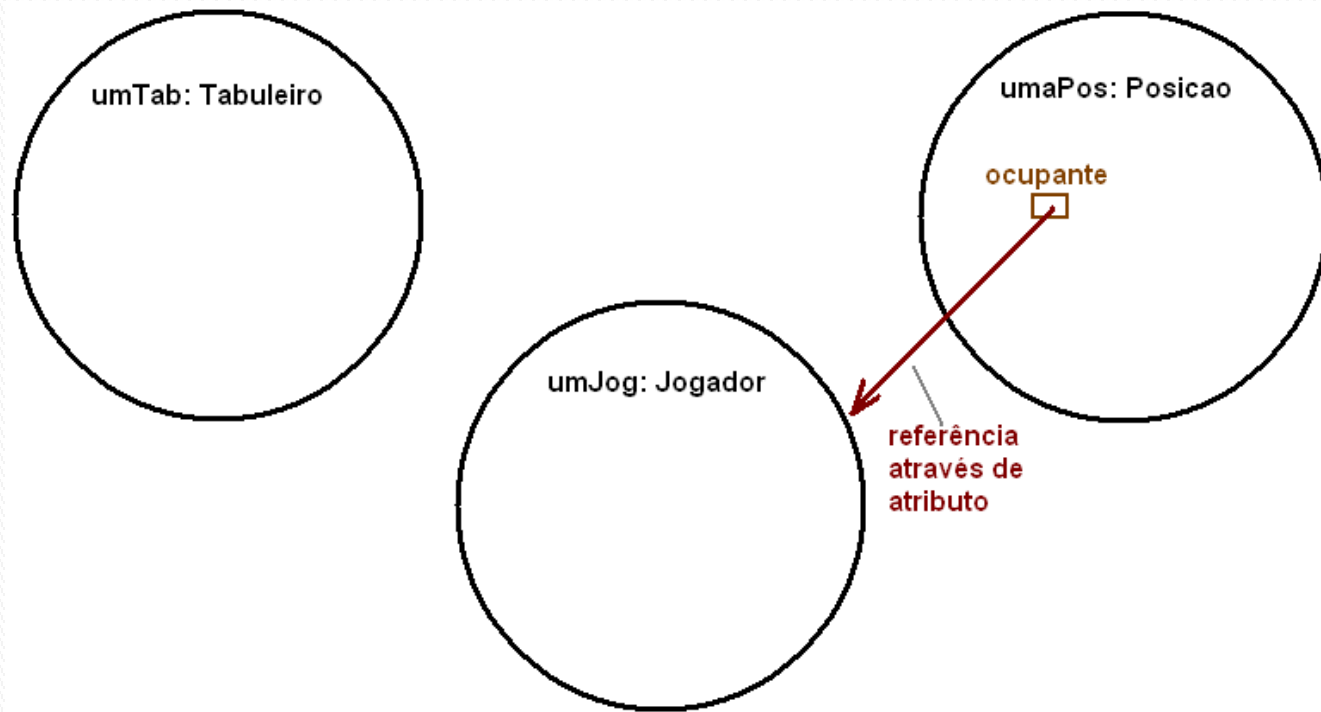
- Posição gerada passa a ser referenciada pelo tabuleiro

2 – Execução do método *alocarPeao*



- Instância de *Tabuleiro* invoca *alocarPeao* de instância de *Posicao*, passando a referência de uma instância de *Jogador*

2 – Execução do método *alocarPeao*



- Instância de *Jogador* passa a ser referenciada pela instância de *Posicao*

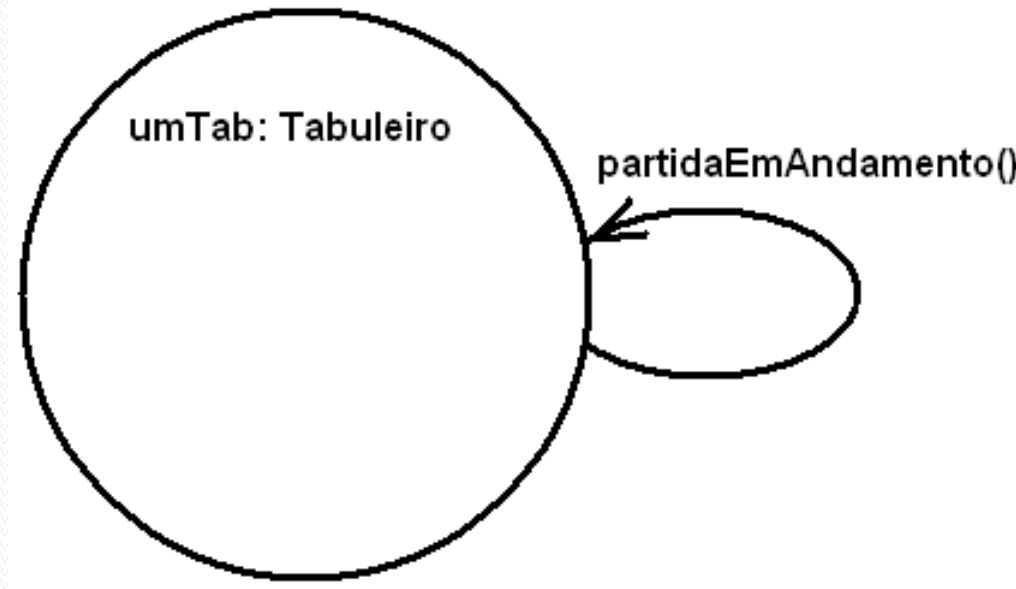
Comportamento observado

- Nos dois exemplos anteriores ocorreram dois tipos distintos de interação
 - No primeiro, um método foi invocado de uma classe
 - No segundo, um método foi invocado de uma instância

Mais um exemplo

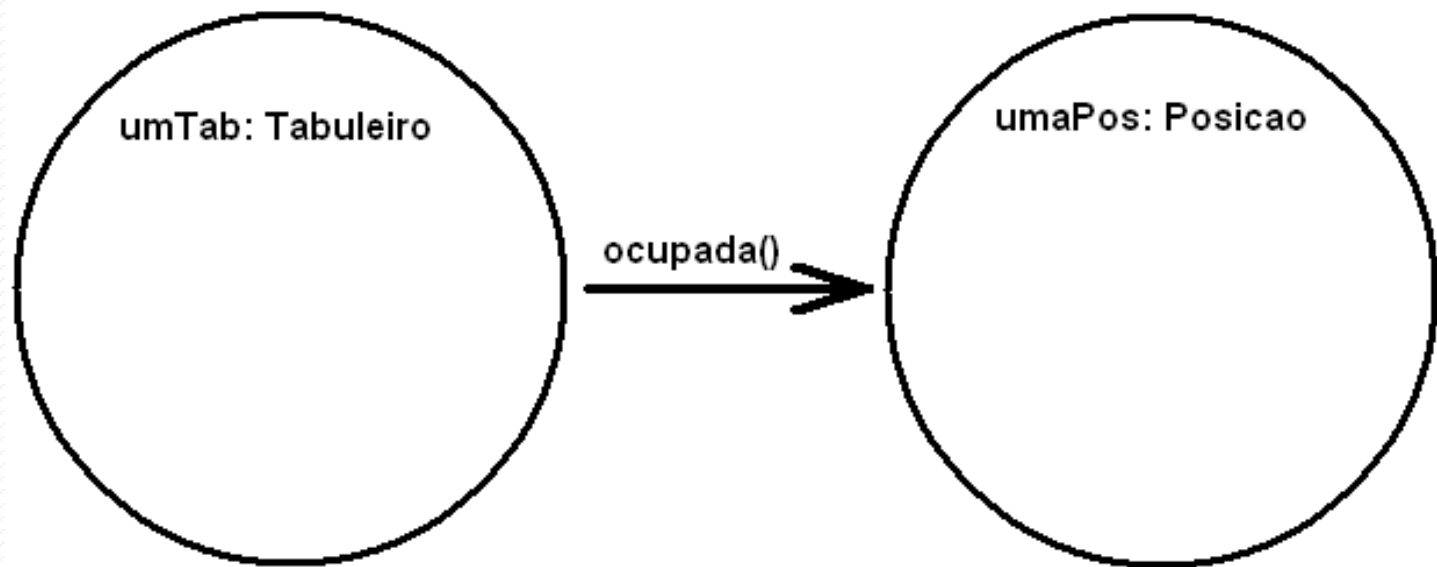
- No exemplo a seguir, um conjunto de interações entre instâncias para que uma tarefa seja executada
 - *Resposta (do tabuleiro) à seleção de uma posição*
 - Observar que a notação adotada é limitada → não mostra invocações que só ocorrem caso uma condição seja verdadeira ou repetição, por exemplo

3 – resposta à seleção de uma posição



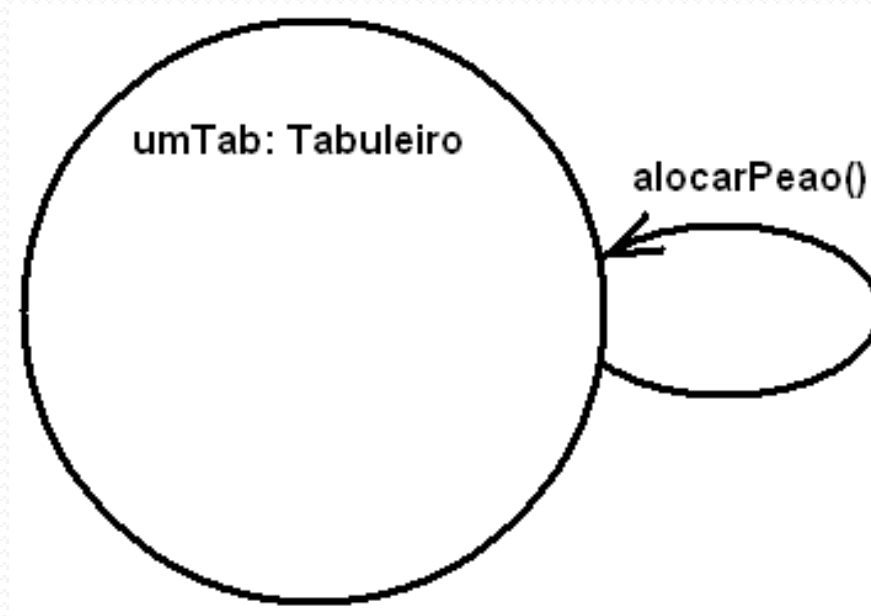
- 1 – Ao perceber que uma posição recebeu um *click* de mouse, tabuleiro verifica se a partida está em andamento
 - Caso contrário, as próximas etapas não ocorrerão

3 – resposta à seleção de uma posição



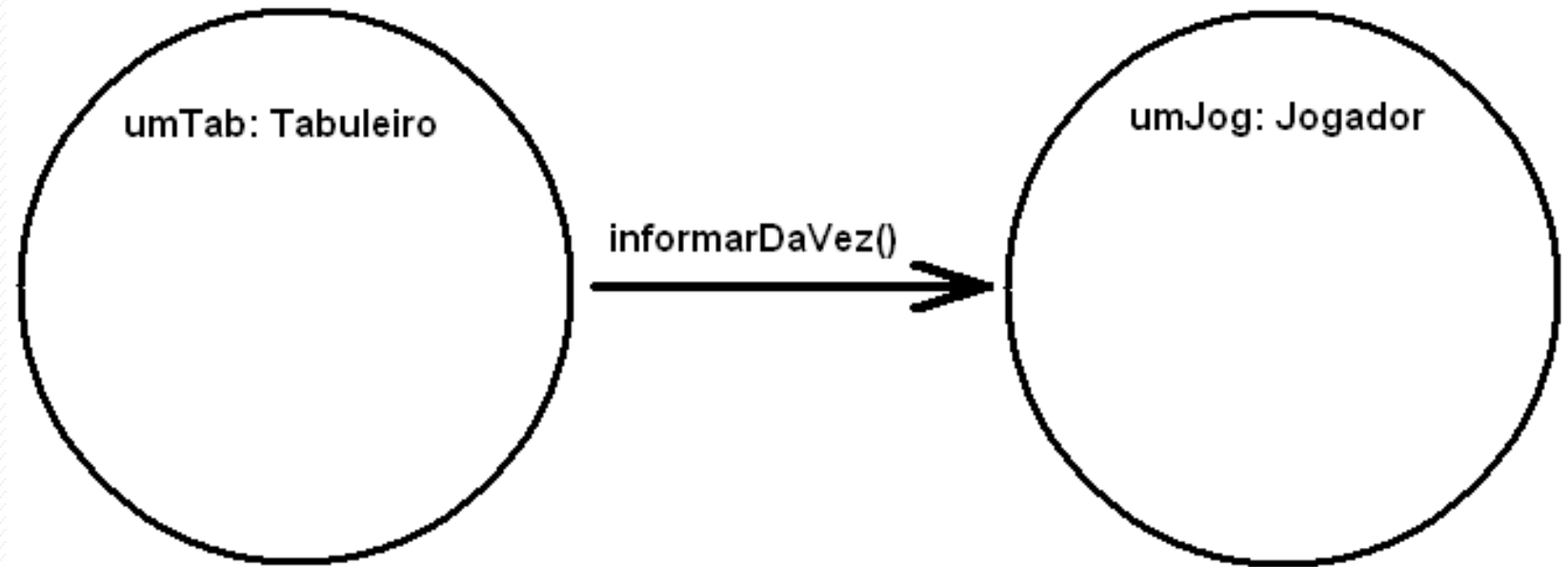
- 2 – Verificado que a partida está em andamento, tabuleiro avalia se a posição selecionada está ocupada
 - Se estiver, as próximas etapas não ocorrerão

3 – resposta à seleção de uma posição



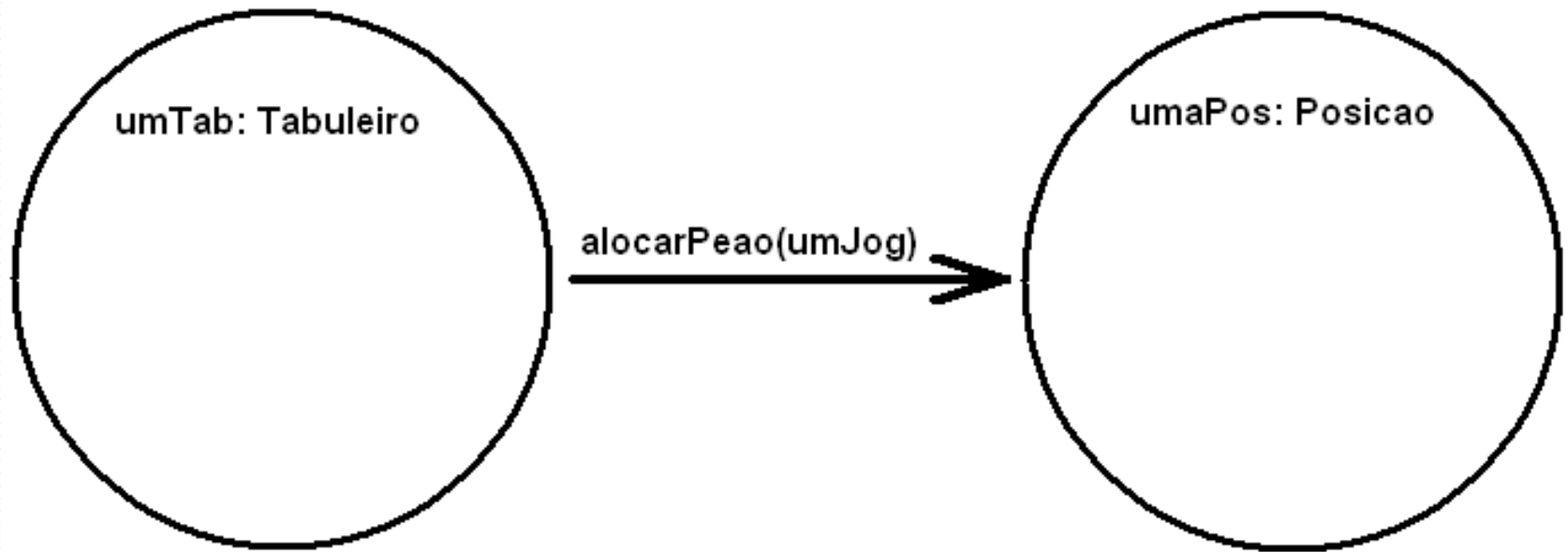
- 3 – Verificado que a partida está em andamento e que a posição selecionada está desocupada, tabuleiro invoca método *alocarPeao*, responsável por posicionar o peão

3 – resposta à seleção de uma posição



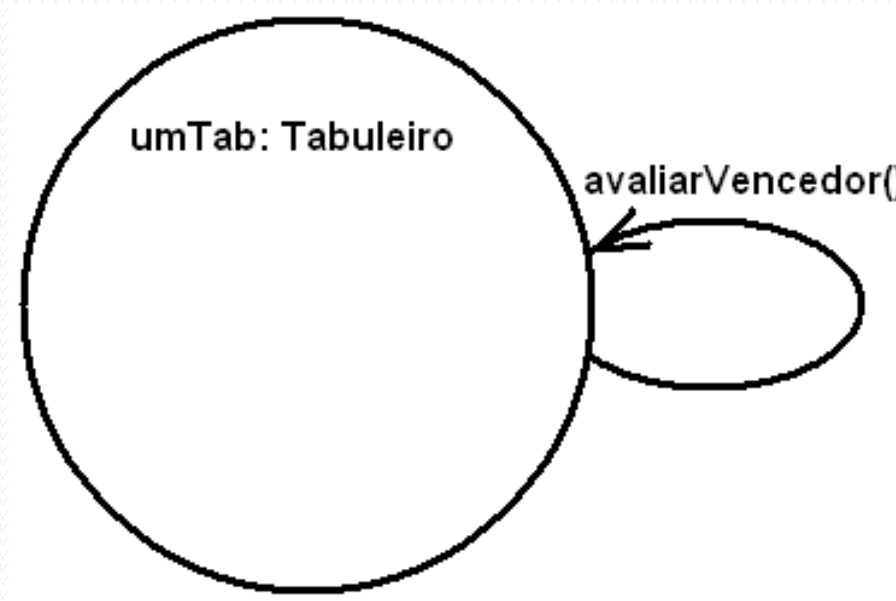
- 4 – Tabuleiro identifica jogador da vez
 - Tabuleiro mantém referência dos dois jogadores (atributo)
 - Invoca de um deles o método *informarDaVez*,
 - Se retorna true ele é o jogador da vez; false, é o outro

3 – resposta à seleção de uma posição



- 5 -Identificado o jogador da vez, tabuleiro invoca o método *alocarPeao* da posição selecionada, passando esse jogador como argumento

3 – resposta à seleção de uma posição



- 6 – Alocado o peão na posição selecionada, tabuleiro invoca seu método *avaliarVencedor*
 - Para verificar se jogador da vez conseguiu ocupar três posições adjacentes, alinhadas

3 – resposta à seleção de uma posição



- 7 – Na execução do método *avaliarVencedor* (passo 6) tabuleiro invoca o método *avaliarSeMesmoJogador* da posição selecionada
 - Para todas as possibilidades (repetição) de três posições adjacentes alinhadas, com a participação da posição selecionada, passando as outras duas posições como argumento

O que os exemplos mostram?

- Frações da execução do programa Jogo-da-velha
 - Em uma notação primitiva
 - Bolas e setas...
 - UML tem mecanismos mais eficazes
- Algo que não é percebido na visão de tempo de desenvolvimento
 - Olhar classes com seus atributos e métodos não proporciona visão do seu comportamento quando o programa estiver em execução

Visões complementares

- Dois pontos de vista do mesmo elemento observado
 - Software em tempo de desenvolvimento
 - Classes com atributos e métodos
 - Software em tempo de execução
 - Instâncias de classe interagindo
- Ambos importantes
 - Para descrever um software
 - Para compreender um software

Projetar um software demanda

- Visão de que partes compõem esse software
 - As classes
 - Visão de tempo de desenvolvimento
- Antever como esse software se comportará quando posto em execução
 - Vislumbrar as várias situações da execução
 - Vislumbrar a interação entre instâncias em cada uma
 - Registrar isso na especificação de projeto
 - Visão de tempo de execução

Considerações sobre esta aula

- Um software pode ser visto como
 - Um conjunto de classes com seus atributos e métodos
 - Um conjunto de objetos interagindo
- Os dois pontos de vista são complementares e importantes e devem ser
 - Tratados durante a produção de software
 - Registrados na especificação de projeto
- UML → instrumento para registrar e vislumbrar os dois pontos de vista

Referências

Booch, G.; Jacobson, I. e Rumbauch, J. **UML: Guia do Usuário**. Campus, 2006.

Silva, R. P. **UML 2 em modelagem orientada a objetos**. Visual Books, 2007.