



Trilha de Programação Orientada a Objetos

OT10 – Construção da casa (movimentar aberturas)

INDICADORES

- Desenvolver capacidades linguísticas de modo a saber usar adequadamente a linguagem oral e escrita em diferentes situações e contextos;
- Conhecer o caráter do conhecimento científico aplicando a metodologia científica e utilizando redação acadêmica na realização da pesquisa, na escolha de métodos, técnicas e instrumentos de pesquisa;
- Compreender e aplicar técnicas de relações humanas visando o desenvolvimento da liderança e relacionamento em equipe, preservando aspectos éticos e organizacionais;
- Gestão das atividades;
- Cumprimento dos prazos;
- Analisa e avalia o funcionamento de computadores e periféricos em ambientes computacionais;
- Codifica programas computacionais utilizando lógica de programação e respeitando boas práticas de programação;
- Utilizar estruturas de dados definindo-as e aplicando-as adequadamente nos programas;
- Desenvolver sistemas em diferentes segmentos;
- Sistematizar o desenvolvimento de software na concepção do projeto de software.

MOVIMENTAR ABERTURAS

Dando continuidade a nossa proposta, faremos agora a funcionalidade “movimentar aberturas” de nossa casa, que corresponde ao “case 1” do método **`exibeMenu()`**, na classe **Controladora**. Relembrando, essa funcionalidade permite que possamos modificar o estado (aberta ou fechada) de uma porta ou janela. Inicialmente, perguntaremos ao usuário qual o tipo de abertura ele deseja movimentar. Na sequência, pediremos que o usuário indique qual porta ou janela especificamente deseja mudar o estado e solicitar que informe também o novo estado. Sabendo qual abertura especificamente o usuário deseja movimentar e seu estado, o último passo é então modificar seu estado efetivamente. Sendo assim, vamos começar as implementações necessárias na classe **Controladora**:

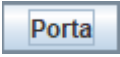

```
case 1:
    String tipoAbertura = EntradaSaida.solicitaTipoAbertura();
    break;
```

Bem, o primeiro passo que precisaremos fazer haja vista a linha implementada, é criar o método **solicitaTipoAbertura()**, onde o usuário informará o que deseja movimentar. Na classe **EntradaSaida**, após o último método criado, implemente:

```
public static String solicitaTipoAbertura() {
    String[] opcoes = {"Porta", "Janela"};

    int tipoAbertura= JOptionPane.showOptionDialog(null, "Informe qual tipo de abertura deseja mover",
        "Mover abertura", JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE,
        null, opcoes, opcoes[0]);

    if(tipoAbertura==0) {
        return "porta";
    }else {
        return "janela";
    }
}
```

O método **solicitaTipoAbertura()** retornará uma **String**, correspondente ao botão do **showOptionDialog()** que foi clicado:  (retorno 0) ou  (retorno 1). De posse dessa informação, saberemos em qual das listas de aberturas da classe **Casa** precisaremos trabalhar. Continue a implementação no “case 1”:

```
case 1:
    String tipoAbertura = EntradaSaida.solicitaTipoAbertura();

    ArrayList<Aberturas> listaDeAberturas = new ArrayList<Aberturas>();

    if(tipoAbertura.equals("porta")) {
        listaDeAberturas = this.casa.getListaDePortas();
    }else {
        listaDeAberturas = this.casa.getListaDeJanelas();
    }

    break;
```

Tratando-se de uma porta a ser movimentada, por exemplo, buscamos na classe **Casa**, pelo método **getListaDePortas()** a lista de portas inteira e a salvamos na **listaDeAberturas**, declarada logo acima de estrutura condicional do tipo **if**. Ignorando a chave destacada, correspondente ao fechamento do **else**, pois ela já foi implementada e serve somente para sua localização, implemente:

```

    }

    int posicao = EntradaSaida.solicitaAberturaMover(listaDeAberturas);
    int novoEstado=0;
    break;

```

O método **solicitaAberturaMover()** que vamos criar em breve na classe **EntradaSaida**, receberá como parâmetro a lista de **Aberturas** que buscamos na classe **Casa**, exibindo as descrições das janelas ou portas, facilitando a escolha do usuário. Na classe **EntradaSaida** então, após o último método criado, implemente, ignorando a numeração de linhas, pois ela servirá para as explicações posteriores somente:

```

59 public static int solicitaAberturaMover(ArrayList<Aberturas> listaDeAberturas) {
60     String tipoAbertura= listaDeAberturas.get(0).getClass().getName();
61     tipoAbertura=tipoAbertura.replaceAll("modelo.", "");
62     int qtdeAbertura= listaDeAberturas.size();
63     String[] descricoesAberturas= new String[qtdeAbertura];
64
65     for(int i=0; i<qtdeAbertura; i++) {
66         descricoesAberturas[i]=listaDeAberturas.get(i).getDescricao();
67     }
68
69     String msg = "Escolha a "+tipoAbertura+" a ser movimentada";
70     JComboBox exibicaoAberturas = new JComboBox(descricoesAberturas);
71     int confirmacao = JOptionPane.showConfirmDialog(null, exibicaoAberturas, msg,
72         JOptionPane.OK_CANCEL_OPTION);
73
74     if(confirmacao==0) {
75         return exibicaoAberturas.getSelectedIndex();
76     }else {
77         return -1;
78     }
79 }

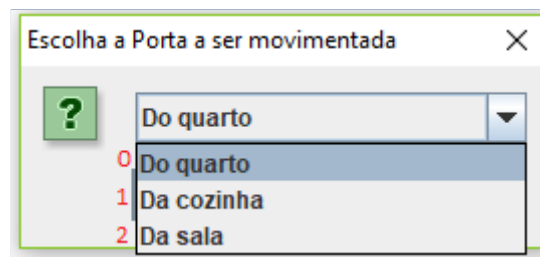
```

Vamos lá! Para que possamos ter somente um método que possa trabalhar tanto com a lista de portas quanto com a lista de janelas, precisamos que o método **solicitaAberturaMover()** receba uma lista do tipo mais genérico. Não confunda genérico nesse caso com a classe **Object** que já discutimos, mas quando falamos em herança, a classe mãe é dada como a classe genérica e as filhas como as classes especializadas. Como iremos receber um tipo genérico, precisamos descobrir, ainda assim, se estamos falando de uma **Porta** ou **Janela**. Na linha **60**, fizemos essa descoberta, obtendo do primeiro elemento da lista (**get(0)**) o nome de sua classe (**getClass().getName()**). O nome da classe é uma **String**, obviamente, que então salvamos em **tipoAbertura**. No entanto, o **getName()** retorna o caminho completo da classe, por isso na linha **61** utilizamos o método **replaceAll()** da classe **String** para que possamos substituir “*modelo.*” por “”, sobrando então somente “**Porta**” ou “**Janela**” que é o que precisamos.

Já na linha **62**, pelo método **size()** do **ArrayList**, que você pesquisou e fichou na OT anterior, descobrimos o tamanho da lista, ou seja, quantos elementos ela possui. Esse dado nos permite então criar um vetor de **Strings** chamado **descricoesAberturas** na linha **63**, desse mesmo tamanho, para que

possamos armazenar as devidas descrições de cada **Porta** ou **Janela** da **listaDePortas** ou da **listaDeJanelas** da classe **Casa**. Na linha 65, declaramos uma estrutura de repetição do tipo **for**, que repetirá **qtdeAbertura** vezes, e preencherá cada uma das posições do vetor **descricoesAberturas**. É importante ressaltar que na posição 0 do vetor, estará a descrição da janela ou porta da posição 0 da lista também.

Nos interessa saber a posição da janela/porta que o usuário deseja movimentar. Sendo assim, é útil que utilizemos um **JComboBox** para exibir tais descrições, pois sabemos que com esse componente, teremos acesso a exatamente qual posição das opções mostradas o usuário selecionou, veja o exemplo:



No exemplo acima, a primeira porta adicionada na casa foi a “Do quarto” e caso o usuário selecione-a, saberemos que sua posição no **ArrayList** é a 0, pois como já mencionado, a ordem exibida no **JComboBox** é a mesma ordem de adição da **listaDePortas** ou da **listaDeJanelas** na classe **Casa**. Continuando, na linha **70** do método **solicitaAberturaMover()**, é criado o objeto **JComboBox** chamado **exibicaoAberturas**, que recebe em seu método construtor o vetor de **Strings descricoesAberturas** com as devidas descrições das portas ou janelas a serem movimentadas. Na linha **71**, a variável **confirmacao** receberá 0 ou 1, correspondente ao usuário ter clicado em “OK” ou “Cancelar”, respectivamente. Caso o usuário clicar em “OK”, significa que ele selecionou uma das aberturas para ser movimentada e precisamos então saber sua posição, por conta disso, é retornada a posição selecionada com o auxílio do método **getSelectedIndex()**, da classe **JComboBox**. No entanto, se o usuário cancelar a ação de escolha de qual abertura deseja movimentar, o método **solicitaAberturaMover()** retornará então **-1**. Dessa forma, a movimentação só será efetivamente realizada se uma posição válida (0 ou superior) for retornada por esse método.

Retorne na classe **Controladora**, ainda dentro do “case 1” e vamos continuar implementando a funcionalidade de movimentar aberturas. O que está destacado é apenas para sua localização e a numeração de linhas deve ser ignorada, implemente:

```

77     int posicao = EntradaSaida.solicitaAberturaMover(listaDeAberturas);
78     int novoEstado=0;
79
80     if(posicao!=-1) {
81         novoEstado = EntradaSaida.solicitaEstado(tipoAbertura);
82         Aberturas abertura = this.casa.retornaAbertura(posicao, tipoAbertura);
83         this.casa.moverAbertura(abertura, novoEstado);
84     }else {
85         EntradaSaida.exibeMsgAbertura();
86     }
87     break;

```

!Na estrutura condicional do tipo **if** da linha 80, estamos verificando se a posição retornada pelo método **solicitaAberturaMover()** é diferente de **-1**. Se sim, sabemos que temos uma abertura a ser movimentada. Portanto, solicitamos um novo estado para ela na linha 81. O método **solicitaEstado()** já foi criado e nesse momento, estamos reutilizando-o. Já na linha 82, temos um método, ainda não criado, que irá retornar o **objeto** porta ou janela, ou seja, a abertura a ser movimentada. Vamos implementar o método **retornaAbertura()** na classe **Casa**. Esse método deve estar na classe **Casa** pois precisa acessar e retornar especificamente um elemento de uma das listas, que são atributos dessa classe. Vamos implementá-lo, após o fechamento do último método:

```

public Aberturas retornaAbertura(int posicao, String tipoAbertura) {
    if(tipoAbertura.equals("porta")) {
        return this.listaDePortas.get(posicao);
    }else {
        return this.listaDeJanelas.get(posicao);
    }
}

```

O método **retornaAbertura()** recebe como parâmetro a posição do objeto a ser retornado e o tipo de abertura, para que seja possível acessar a lista correta, retornando o **objeto** daquela posição. Recebemos na linha 82 o objeto do tipo **Aberturas** retornado pelo **retornaAbertura()**. Passamos então como argumento para o método **moverAbertura()** na linha 83 respectivamente o objeto (referência) e seu novo estado. Vamos criar agora, na classe **Casa**, após o fechamento do último método o método **moverAbertura()**:

```

public void moverAbertura(Aberturas abertura, int novoEstado) {
    abertura.setEstado(novoEstado);
}

```

O método **moverAbertura()** receberá a referência do objeto a ser movimentado e seu novo estado, como já mencionado. Sua implementação é simples, setando o novo estado para o objeto **Aberturas (Porta ou Janela)** recebido. No entanto, retomando, esse processo acontecerá somente se na linha 80 a condição for verdadeira, caso não seja, sabemos que nenhuma abertura deverá ser movimentada. Relembrando (pois você já implementou o código a seguir):

```

80     if(posicao!= -1) {
81         novoEstado = EntradaSaida.solicitaEstado(tipoAbertura);
82         Aberturas abertura = this.casa.retornaAbertura(posicao, tipoAbertura);
83         this.casa.moverAbertura(abertura, novoEstado);
84     } else {
85         EntradaSaida.exibeMsgAbertura();
86     }

```

Precisamos então, para finalizar, criar o método **exibeMsgAbertura()**, na classe **EntradaSaida**. Após o último método lá criado, implemente:

```

public static void exibeMsgAbertura() {
    JOptionPane.showMessageDialog(null, "Nenhuma abertura será movimentada");
}

```

Sim, o método é bastante simples e apenas exibe uma mensagem, informando ao usuário que nenhuma abertura será movimentada. Para finalizarmos, convidamos você a implementar uma linha adicional, para que possamos verificar se realmente a porta ou janela teve seu estado alterado, ou seja, se foi movimentada. Para isso, volte na classe **Controladora**, no “case 1” e implemente a linha destacada, no local indicado:

```

this.casa.moverAbertura(abertura, novoEstado);
System.out.println("Novo estado da "+tipoAbertura+": "+abertura.getEstado());
} else {

```

E agora, **teste sua aplicação**, crie a casa, crie pelo menos 2 portas e 2 janelas e verifique se toda a implementação que fizemos nessa OT está ok!

TIPOS PRIMITIVOS E POR REFERÊNCIA

Em Java temos os tipos primitivos de dados e os tipos por referência. Os tipos primitivos você conheceu e utilizou na trilha de lógica de programação: *int*, *double*, *boolean*, *float* (...). Já os tipos por referência você passou a conhecer mais profundamente nessa trilha em que está, pois nada mais são do que os **objetos** que são do tipo de uma determinada classe, sendo uma instância (cópia) dela, permitindo então acesso aos métodos e atributos que lá estão definidos.

Um tipo primitivo é capaz de armazenar somente um valor. Já um tipo por referência pode apontar para outros tipos por referência e outros tipos primitivos. Uma outra questão que diferencia um tipo primitivo de um tipo por referência é a sua inicialização. Quando declaramos uma variável de um tipo primitivo, existem duas situações onde sua **inicialização** difere:

- Quando se trata de uma variável local (dentro de um método), sua inicialização não é automática, ou seja, até que um valor seja atribuído a essa variável, ela permanece sem valor algum;
- Quando se trata de uma variável global, ou seja, um atributo em uma classe, sua inicialização é automática, ou seja, recebe um valor padrão

até que seja atribuído o valor necessário. Para os tipos primitivos numéricos: float, double, int o valor de inicialização é 0, por exemplo.

Porém, quando declaramos uma variável de tipo por referência, ou seja, um **objeto**, a inicialização das variáveis de tipos primitivos ou dos outros objetos para os quais ele aponta (seus atributos), podem ter valores diferentes. Podemos definir no método **construtor** de cada classe quais valores seus atributos devem assumir no momento da instância do objeto.

Além disso, quando estamos nos referindo a um **objeto**, sempre que o utilizamos em qualquer parte do código, estamos trabalhando com a mesma referência dele, ou seja, sempre com o **mesmo objeto**. Você pôde perceber isso no método **moverAbertura()**, onde repassamos a referência do objeto **Aberturas** que queremos movimentar e o método faz a alteração do estado do objeto unicamente, não em uma cópia dele. É por isso que não precisamos substituir na **listaDePortas** ou na **listaDeJanelas** da **Casa** um novo objeto **Porta** ou **Janela**. Em outras palavras, por conseguirmos acessar diretamente a referência do objeto, estamos alterando diretamente ele.

Até a próxima e última OT dessa segunda etapa da trilha de programação orientada a objetos 😊 !