



# Trilha de Programação Orientada a Objetos

## OT8 – Construção da casa (explicações iniciais, estrutura, menu)

---

### INDICADORES

- Desenvolver capacidades linguísticas de modo a saber usar adequadamente a linguagem oral e escrita em diferentes situações e contextos;
- Conhecer o carácter do conhecimento científico aplicando a metodologia científica e utilizando redação académica 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.

---

### FICHAMENTO

#### Classe JComboBox

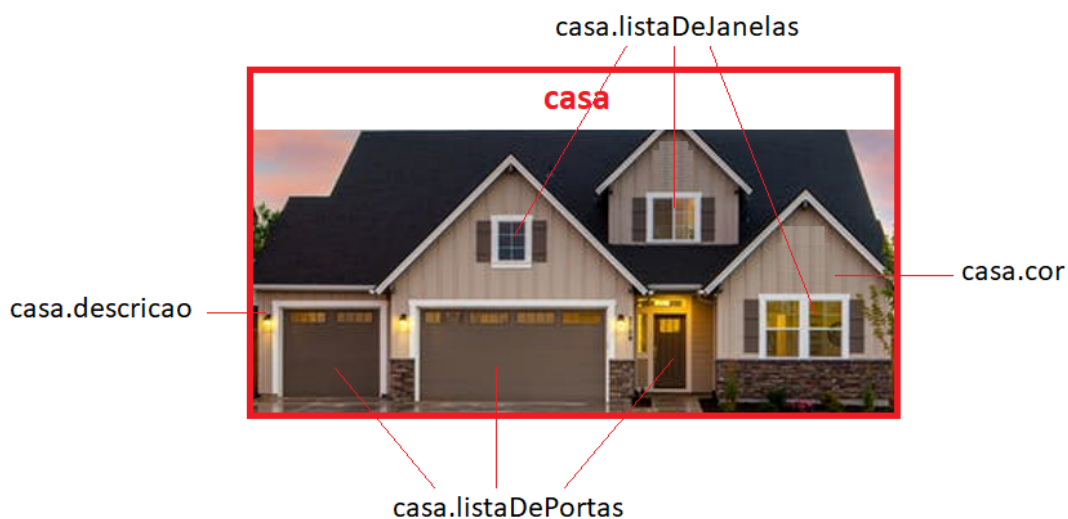
- Definição (utilidade)
- Método construtor JComboBox(E[] items)
- Método `getSelectedIndex()`
- Método `getSelectedItem()`

Use a referência da documentação oficial no site da (<https://docs.oracle.com/en/>) para seu fichamento, porém, procure utilizar outra fonte confiável também como 2ª citação, haja vista que na documentação oficial o conteúdo é bastante técnico.

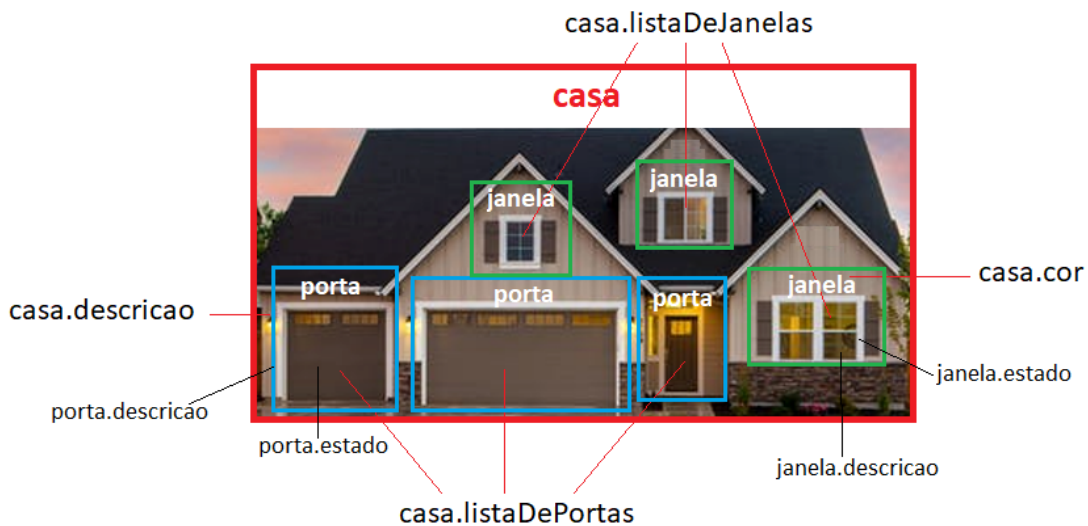
## EXPLICAÇÕES INICIAIS

Muito bem! Se você chegou nesse documento é porque acabou de concluir a primeira parte da trilha de programação orientada a objetos, que se trata do desenvolvimento da Calculadora. Na primeira etapa, você aprendeu na prática como os principais conceitos desse paradigma de programação podem/devem ser aplicados, e mesmo se tratando de uma proposta de baixa complexidade, sempre lhe instigam a pensar a utilização de tais conceitos em aplicações mais robustas e escalonáveis. Nesta segunda etapa, a que começaremos agora, também demonstraremos alguns conceitos e novas aplicações, mas de um modo diferente, que explicaremos a seguir. Na terceira etapa dessa trilha, você fará 03 atividades práticas, que nada mais são do que o desenvolvimento de um *software* para cada atividade, aplicando os conceitos de POO até então aprendidos (e novos, se necessários).

Abordando então a segunda etapa da trilha, faremos a construção de uma casa. Não vamos ter aula no laboratório de edificações, fique tranquilo(a)! Desenvolveremos a simulação da construção de uma casa, que contém uma descrição, cor, uma lista de portas e uma lista de janelas. Se explicássemos por meio de um desenho, seria algo como:



Então, temos um objeto do tipo **Casa**, que contém seus atributos: **descricao**, **cor**, **listaDePortas** e **listaDeJanelas**. No entanto, a lista de portas é composta por objetos do tipo **Porta** e a lista de janelas, por objetos do tipo **Janela**. Da mesma forma, se representássemos, seria algo como:



Cada porta e janela possuem, por sua vez, os atributos **descricao** e **estado** (que pode ser *aberta* ou *fechada*).

Essa proposta, pode-se dizer, é um pouco mais lúdica do que a Calculadora, e tem como objetivos:

- Reforçar os conceitos e aplicações de POO já trabalhados (porém, nem todos serão abordados novamente);
- Reforçar o padrão de projetos MVC;
- Apresentar o conceito de listas, que são muito úteis na programação (e parecidas com vetores);
- Subsidiar o desenvolvimento das 03 atividades práticas que você desenvolverá na terceira etapa.

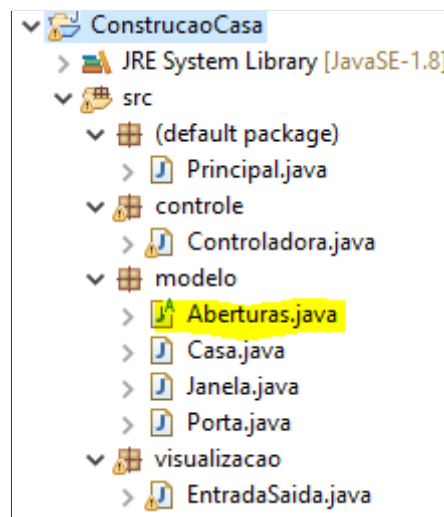
## ESTRUTURA

Então, vamos lá! O primeiro passo é a criação do projeto, que novamente, será um Java Project. Você poderá criar o projeto na mesma *workspace* que vinha desenvolvendo as versões da Calculadora, ou então, criar uma nova, como preferir. O caminho, você já conhece: **File >> New >> Java**

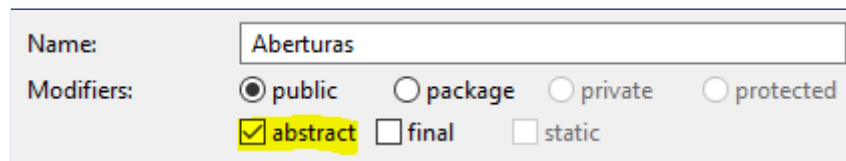
## Project:

The screenshot shows the 'New Java Project' dialog box. The 'Project name' field is filled with 'ConstrucaoCasa'. The 'Use default location' checkbox is checked, and the 'Location' field shows the path 'C:\Users\Indianara\Desktop\ConstrucaoCasaEdicoes\ConstrucaoCasa'. Under the 'JRE' section, the 'Use an execution environment JRE' radio button is selected, and the dropdown menu shows 'JavaSE-1.8'. The 'Project layout' section has the 'Create separate folders for sources and class files' radio button selected. The 'Working sets' section has the 'Add project to working sets' checkbox unchecked. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

\*\* Ignore o *location* da imagem, mas você deve manter selecionada a opção “*use default location*” para que o projeto seja criado em sua *workspace* atual. Criado o projeto, agora você deverá criar a estrutura de pacotes e classes a seguir, mas antes, **leia as duas orientações** que sucedem o *print* da estrutura:



**Orientação 1:** A classe em destaque trata-se de uma classe abstrata, então, ao criá-la, você pode marcar a opção...



... ou então, criá-la normalmente e adicionar a palavra *abstract* em sua declaração:

```
package modelo;  
  
public abstract class Aberturas {
```

**Orientação 2:** Na classe **Principal**, você deverá marcar a opção de criação do método **main()** (essa sim, é mais fácil marcar do que digitar 😊) e veja que essa classe foi criada no pacote **default**. Esse pacote **não** deve ser criado, você deverá criá-la diretamente na raiz do seu projeto, na pasta **src**.

Criada a estrutura do projeto, vamos falar um pouco sobre as classes. Como já mencionado, um dos objetivos dessa proposta é o reforço do padrão de projetos MVC e sendo assim, os pacotes/classes criados já sugerem a divisão de responsabilidades pregada por esse modelo:

- **default:** nesse pacote, temos somente a classe **Principal**, responsável pela inicialização do *software*, por conter o método **main()**. Haja vista que essa classe tem sua função bem específica e que não se encaixa nas responsabilidades das demais camadas do MVC, ela ficará na raiz do projeto;
- **controle:** nesse pacote, temos somente a classe **Controladora**, que em função semelhante a classe de mesmo nome na proposta da Calculadora, será a classe responsável por receber as solicitações, encaminhar as classes responsáveis por atendê-las, receber os retornos e continuar nesse ciclo, até que o programa seja encerrado. Essa classe funciona como uma ponte entre as classes do pacote **modelo** e as classes do pacote **visualizacao**.
- **modelo:** em nossa proposta, como mostrado nas ilustrações, criaremos uma **Casa**, que possui **Portas** e **Janelas**. Tanto a **Casa** quanto suas **Aberturas** possuem uma estrutura bem definida e sendo assim, é interessante que sejam modeladas em uma classe. A classe abstrata (mãe) **Aberturas** servirá de referência para as classes **Porta** e **Janela**, uma vez que ambas possuem as mesmas características (descrição e estado), não sendo assim necessário que códigos sejam repetidos em ambas.
- **visualizacao:** assim como no pacote **controle**, apenas uma classe ficará nesse pacote, a classe **EntradaSaida**. Ela será responsável por todas as entradas e saídas de dados ao usuário, a exemplo da

classe **EntradaSaida** que também tínhamos na proposta da Calculadora.

## MENU

Vamos; iniciar então o desenvolvimento de nossa proposta da construção da casa, pelo menu, que ficará na classe **Controladora**. Implemente:

```
package controle;
import javax.swing.JOptionPane;
import visualizacao.EntradaSaida;
import modelo.*;

public class Controladora {

    private Casa casa = null;

    public void exibeMenu() {
        int opcao;
        do {
            opcao = EntradaSaida.solicitaOpcao();

            switch(opcao){
                case 0:
                    JOptionPane.showMessageDialog(null, "Construir casa");
                    break;

                case 1:
                    JOptionPane.showMessageDialog(null, "Movimentar portas ou janelas");
                    break;

                case 2:
                    JOptionPane.showMessageDialog(null, "Ver informações da casa");
                    break;
            }
        }while(opcao!=3);

        EntradaSaida.exibeMsgEncerraPrograma();

        System.exit(0);
    }
}
```

As linhas destacadas são para que você não esqueça dos *imports* necessários (mais alguns serão feitos adiante), do atributo *casa*, e a delimitação do método **exibeMenu()**. Ainda, você deve estar com 02 erros nesse momento:

- na chamada do método **EntradaSaida.solicitaOpcao();**
- na chamada do método **EntradaSaida.exibeMsgEncerraPrograma()**

Os métodos invocados não existem e esse é o motivo dos erros, vamos criá-los então na classe **EntradaSaida**. Implemente:

```

package visualizacao;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;

public class EntradaSaida {

    public static int solicitaOpcao() {
        String[] opcoes= {"Construir casa", "Movimentar portas ou janelas",
            "Ver informações da casa", "Sair do programa"};
        JComboBox<String> menu = new JComboBox<String>(opcoes);
        JOptionPane.showConfirmDialog(null, menu, "Selecione a opção desejada",
            JOptionPane.OK_CANCEL_OPTION);
        return menu.getSelectedIndex();
    }

    public static void exibeMsgEncerraPrograma() {
        JOptionPane.showMessageDialog(null, "O programa será encerrado!");
    }

}

```

No método **solicitaOpcao()**, estamos utilizando uma nova classe do pacote **swing** (mesmo pacote de componentes visuais da classe **JOptionPane**) que é a classe **JComboBox**. No início dessa OT, solicitamos que você efetuasse o fichamento sobre essa classe e lá, pedimos que você falasse sobre a utilização dessa classe e sobre um de seus métodos construtores, que é o que vamos utilizar. Falando nisso, você lembra o que é e para que serve um método construtor? **Se não, revise seus fichamentos e a OT em que tratamos desse conceito.**

Pois bem, o método construtor da classe **JComboBox** sobre a qual solicitamos o fichamento é o método que recebe um **array** de itens. Esses itens, podem ser de vários tipos diferentes. No entanto, em nosso caso, precisamos exibir as opções do menu, que por sua vez são do tipo **String**, por isso, devemos especificar na criação desse componente que gostaríamos que ele exibisse objetos do tipo **String**. Sem essa especificação, nosso **JComboBox** funcionaria da mesma forma, no entanto, como o Java é uma linguagem fortemente tipada, é interessante que quando saibamos os tipos com os quais estamos lidando (e eles não venham a mudar) que nós os indiquemos explicitamente, tornando nosso código mais eficiente. Conversaremos novamente sobre essas questões mais adiante, quando falarmos sobre a classe **ArrayList**.

Com base em seu fichamento, você conseguiria explicar o motivo do retorno do método **solicitaOpcao()** ser um valor inteiro e a lógica do **switch-case** aplicada no método **exibeMenu()**? **Discuta com seu orientador no momento da validação e teste sua aplicação.** Por enquanto, ainda não desenvolvemos nenhuma funcionalidade em nossa proposta, fizemos apenas o esqueleto dela e alguns métodos iniciais. Inclusive, para cada opção do menu, inserimos apenas uma mensagem, indicando a funcionalidade a qual o usuário terá acesso. Agora, na classe **Principal**, faça a chamada ao método **exibeMenu()** da classe **Controladora**:

```
public static void main(String[] args) {  
    Controladora controladora = new Controladora();  
    controladora.exibeMenu();  
}
```

**Teste sua aplicação.** Na próxima OT, faremos a “construção da casa”, mais precisamente a primeira opção do menu, o “Construir casa”, efetivamente.

Até lá! 😊