Trilha de Programação Orientada a Objetos

OT11 – Construção da casa (ver informações da casa e atividades)

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.

INFORMAÇÕES DA CASA

Chegamos então na última OT da proposta da construção da casa, onde desenvolveremos a última funcionalidade de nosso menu: ver informações da casa. Nessa opção, exibiremos ao usuário os dados dos atributos de nosso objeto do tipo **Casa**: sua **descrição**, sua **cor**, a **descrição** e **estado** de cada porta da **lista de portas** e cada janela da **lista de janelas**, respectivamente.

O primeiro passo é criarmos na classe **Casa** um método que acesse os atributos e gere de forma organizada uma **String** para que possamos exibí-los. Não iremos exibir as informações já nesse método, apenas retorná-las, pois essa não é uma tarefa dessa classe e sim, da classe **EntradaSaida**. Sendo assim, na classe **Casa**, após o último método declarado, implemente o método ***geraInfoCasa()***:

```
public String geraInfoCasa() {
    String informacoes="Descrição: "+this.descricao+"\nCor: "+this.cor+"\nLista de portas:\n";

    for(Aberturas abertura:this.listaDePortas) {
        int estado = abertura.getEstado();
        informacoes+=abertura.getDescricao()+" Estado: "+abertura.getEstado()+"\n";
    }

    informacoes+="\nLista de janelas:\n";

    for(Aberturas abertura:this.listaDeJanelas) {
        int estado = abertura.getEstado();
        informacoes+=abertura.getDescricao()+" Estado: "+abertura.getEstado()+"\n";
    }
    return informacoes;
}
```

Após a criação desse método, podemos invocá-lo em nosso método ***exibeMenu()*** na classe **Controladora**, no "*case 2*", correspondente a funcionalidade que estamos desenvolvendo:

```
case 2:
    String informacoes=this.casa.geraInfoCasa();
break;
```

No entanto, como já mencionado, o ***geraInfoCasa()*** não exibe as informações ao usuário, vamos então desenvolvê-lo. Implemente na classe **EntradaSaida,** após o último método criado:

```
public static void exibeInfoCasa(String informacoes) {
    JOptionPane.showMessageDialog(null, informacoes, "Informações da casa",
            JOptionPane.INFORMATION_MESSAGE);
}
```

Agora, precisamos invocar o método ***exibeInfoCasa()*** logo após a chamada do método ***geraInfoCasa()***. Implemente na classe **Controladora**:

```
case 2:
    String informacoes=this.casa.geraInfoCasa();
    EntradaSaida.exibeInfoCasa(informacoes);
break;
```

Com a chamada do método ***exibeInfoCasa()*** finalizamos a proposta da construção da casa 👏👏👏. No entanto, temos algumas atividades para você desenvolver. Vamos lá?

ATIVIDADES

1. Faça uma validação de dados para que as opções do menu possam ser acessadas somente após a casa ter sido construída. Pense bem na premissa que possuímos para saber se a casa já foi construída ou não;
2. Faça uma validação de dados para que não possam ser informadas quantidades negativas de portas ou janelas ou zero;
3. Faça uma validação para que ao invés de 0 ou 1, sejam exibidos os estados "fechada" ou "aberta" respectivamente no método ***geraInfoCasa()*** para as portas e janelas;

4. Faça o diagrama de classes de nossa proposta na ferramenta Draw.io, que utilizamos na proposta da Calculadora. Pedimos que essa atividade seja feita somente após a validação das atividades anteriores. Uma dica: minimize todos o métodos nas classes para facilitar:

EntradaSaida.java

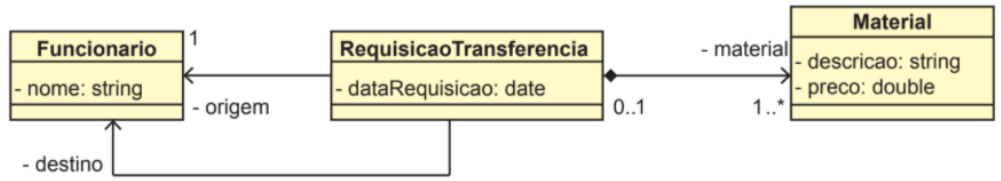
```
1  package visualizacao;
2  import java.util.ArrayList;
7
8  public class EntradaSaida {
9
10     public static int solicitaOpcao() {
16
17     public static void exibeMsgEncerraPrograma() {
20
21     public static String solicitaDescricao(String descricao, int ordem) {
28
29     public static String solicitaCor() {
32
33     public static int solicitaQtdeAberturas(String abertura) {
36
37     public static int solicitaEstado(String tipoAbertura) {
44
45     public static String solicitaTipoAbertura() {
58
59     public static int solicitaAberturaMover(ArrayList<Aberturas> listaDeAberturas) {
80
81     public static void exibeInfoCasa(String informacoes) {
85
86     public static void exibeMsgAbertura() {
89 }
```

Questões do ENADE

Apresente a resposta que julgar correta no momento da validação.

(ENADE 2017)

O seguinte diagrama de classe representa a modelagem de um serviço de transferência de materiais entre funcionários de uma empresa.



Considerando o diagrama de classe apresentado, avalie as afirmações a seguir.

I. A classe Funcionario é abstrata.

II. Uma RequisicaoTransferencia só existe se estiver vinculada a um Material.

III. A classe Funcionario, em razão de sua associação, possui um atributo do tipo RequisicaoTransferencia.

IV. Em uma implementação da classe RequisicaoTransferencia, é necessário adicionar um atributo simples do tipo Material.

É correto o que se afirma em

A II, apenas.
B I e II, apenas.
C III e IV, apenas.
D I, III e IV, apenas.
E I, II, III e IV.

(ENADE 2017)

A área de desenvolvimento de *software* está se tornando cada vez mais complexa. Para lidar com essa realidade, os desenvolvedores contam com linguagens de programação baseadas no paradigma de orientação a objetos, cujos pilares são abstração, encapsulamento, herança e polimorfismo. No código a seguir, observa-se a implementação de classes relacionadas.

```
public abstract class Impressora {
   String nome;

   Impressora() {}

   Impressora(String n) {
      this.nome = n;
   }

   public void imprimir() {}
}

public class Laser extends Impressora {

   public Laser() {}

   public void imprimir() {
      System.out.println("Imprimindo na Laser");
   }
}

class Matricial extends Impressora {

   public Matricial () {}

   public void imprimir() {
      System.out.println("Imprimindo na Matricial");
   }
}
```

```
public class JatoDeTinta extends Impressora {

   public JatoDeTinta() {}

   public void imprimir(){
      System.out.println("Imprimindo na Jato de tinta");
   }

}

public class Main {

   public static void main(String args[]) {
      Impressora imp[] = new Impressora[3];
      imp[0] = new Laser();
      imp[1] = new JatoDeTinta();
      imp[2] = new Matricial();

      for(int i = imp.length - 1; i >= 0; i--){
         imp[i].imprimir();
      }
   }
}
```

Com base nas informações do texto e no código apresentado, avalie as afirmações a seguir.

I. A execução do código, via classe `Main`, resulta na seguinte saída:

Imprimindo na Laser
Imprimindo na Matricial
Imprimindo na Jato de tinta

II. O código faz uso da técnica denominada polimorfismo.

III. O código não será compilado, pois o vetor `imp` foi instanciado por meio da classe abstrata `Impressora`.

É correto o que se afirma em

A) II, apenas
B) II e III, apenas
C) I e III, apenas
D) I, II,III
E) I, apenas