Título da Atividade

PRÁTICA DE POO - IMPLMENTAÇÃO NO PI - PARTE 1

Nome do professor:

Carlos Veríssimo

Nome do Aluno:

CARLOS EDUARDO CHIQUESI ALMEIDA

LUCAS HENRIQUE SILVEIREA FARIAS

LUIS CARLOS TELES SANTOS

JANDERSON BORGES NOGUEIRA BRANCO

PEDRO PAULO DA SILVEIRA CHAVES

Nome da disciplina:

PROGRAMAÇAO ORIENTADA A OBJETOS

Model (Modelo):

- `Veiculo`: Representa o modelo de veículo.

- `Carro`, `Caminhao`, `Moto`: Subclasses de `Veiculo` representando tipos específicos de veículos.

- `Combustivel`: Representa o modelo de combustível.

- `Consumo`: Representa o modelo de consumo.

View (Visão):

- A interface gráfica Swing (`JFrame`, `JPanel`, `JLabel`, `JTextField`, `JButton`, `JTextArea`) é a visualização.

- A lógica relacionada à interface gráfica no método `criarInterface()` é a camada de visão.

Controller (Controlador):

- `GestaoDeCombustivel`: Contém o método `main` e funciona como o controlador.

- Manipula eventos de botão (`cadastrarButton` e `exibirButton`).

- Responsável por processar a entrada do usuário, criar instâncias de veículos, e atualizar a visualização.

Model (model package):

```
// Veiculo.java

import java.util.Date;
```

```
public abstract class Veiculo {
    // ... (código existente)

    public abstract double getDistanciaPercorrida();

    public abstract double getConsumo();

    // ... (restante do código existente)
}

// Carro.java
public class Carro extends Veiculo {
    // ... (código existente)
}

// Caminhao.java
public class Caminhao extends Veiculo {
    // ... (código existente)
}

// Moto.java
public class Moto extends Veiculo {
    // ... (código existente)
}

// Consumo.java
public class Consumo {
    private String placaVeiculo;
    private double valorCombustivelUtilizado;
```

```
    public Consumo(String placaVeiculo, double valorCombustivelUtilizado) {
        this.placaVeiculo = placaVeiculo;
        this.valorCombustivelUtilizado = valorCombustivelUtilizado;
    }

    public double getValorCombustivelUtilizado() {
        return valorCombustivelUtilizado;
    }
}

// Combustivel.java
public class Combustivel {
    private String tipo;

    public Combustivel(String tipo) {
        this.tipo = tipo;
    }

    public String getTipo() {
        return tipo;
    }
}
```

View (view package):

```java
// GestaoDeCombustivelUI.java
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

public class GestaoDeCombustivelUI {
    private JFrame frame;
    private JPanel mainPanel;
    private JTextField tipoField;
    private JTextField modeloField;
    private JTextField placaField;
    private JTextField capacidadeField;
    private JTextField combustivelInicialField;
    private JTextField distanciaPercorridaField;
    private JTextField dataAbastecimentoField;
    private JTextField tipoCombustivelField;
    private JTextField valorAbastecidoField;
    private JButton cadastrarButton;
    private JTextArea resultadoArea;

    public GestaoDeCombustivelUI() {
        createUI();
    }

    private void createUI() {
        frame = new JFrame("Gestão de Combustível");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    mainPanel = new JPanel();
    mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));

    // ... (restante do código da interface gráfica)
}

// Getters para os campos da interface (para serem usados pelo controlador)
public String getTipo() {
    return tipoField.getText().toLowerCase();
}

public String getModelo() {
    return modeloField.getText();
}

// ... (restantes dos getters)

public void setCadastrarButtonListener(ActionListener listener) {
    cadastrarButton.addActionListener(listener);
}

public void setExibirButtonListener(ActionListener listener) {
    // Defina um botão de exibição se necessário
}

public void updateResultadoArea(String result) {
    resultadoArea.setText(result);
}
```

```
    public void display() {
        frame.add(mainPanel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Controller (controller package):

```java
// GestaoDeCombustivelController.java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.List;

public class GestaoDeCombustivelController {
    private GestaoDeCombustivelUI view;
    private List<Veiculo> veiculos;

    public GestaoDeCombustivelController(GestaoDeCombustivelUI view, List<Veiculo> veiculos) {
        this.view = view;
        this.veiculos = veiculos;
```

```
        // Configuração de listeners
        view.setCadastrarButtonListener(new CadastrarButtonListener());
        view.setExibirButtonListener(new ExibirButtonListener());
    }

    // Listener para o botão de cadastro
    private class CadastrarButtonListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                // Obtenha os dados da interface
                String tipo = view.getTipo();
                String modelo = view.getModelo();
                // ... (obtenha os outros campos)

                // Crie uma instância de SimpleDateFormat
                SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
                Date dataAbastecimento = sdf.parse(view.getDataAbastecimento());

                // Crie instâncias apropriadas com base no tipo
                Combustivel combustivel = new Combustivel(view.getTipoCombustivel());
                Veiculo veiculo;

                switch (tipo) {
                    case "carro":
                        veiculo = new Carro(modelo, placa, capacidade, combustivelInicial,
dataAbastecimento, combustivel, valorAbastecido, distanciaPercorrida);
                        break;
                    case "caminhao":
```

```
                veiculo = new Caminhao(modelo, placa, capacidade, combustivelInicial,
dataAbastecimento, combustivel, valorAbastecido, distanciaPercorrida);
                break;
            case "moto":
                veiculo = new Moto(modelo, placa, capacidade, combustivelInicial,
dataAbastecimento, combustivel, valorAbastecido, distanciaPercorrida);
                break;
            default:
                // ... (trate outros tipos, se necessário)
                return;
        }

        veiculos.add(veiculo);

        // ... (limpe os campos da interface)

        view.updateResultadoArea("Veículo cadastrado: " + veiculo + "\n");
      } catch (ParseException ex) {
        // ... (trate exceção de formato de data inválido)
      }
    }
  }

  // Listener para o botão de exibição
  private class ExibirButtonListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
      // ... (código para exibir veículos cadastrados)
    }
  }
```

}

```

#### Main (Main.java):

```java
import javax.swing.*;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            // Crie a lista de veículos
            List<Veiculo> veiculos = new ArrayList<>();

            //
```

Encapsulamento:

No código, a classe `Veiculo` é um exemplo de encapsulamento. Os atributos da classe (`tipo`, `modelo`, `placa`, `capacidade`, `combustivelInicial`, `dataAbastecimento`, `combustivel`, `valorAbastecido`) são privados (`private`), e o acesso a esses atributos é feito por meio de métodos públicos (`public`). Por exemplo, o método `getDistanciaPercorrida()` é público e fornece acesso controlado ao atributo `distanciaPercorrida`.

```java
public abstract class Veiculo {
    private String tipo;
    private String modelo;
    // ...
```
```

```
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    // ...
}
```

Classes:

```
class Carro extends Veiculo {
    private double distanciaPercorrida;

    public Carro(String modelo, String placa, double capacidade, double combustivelInicial, Date dataAbastecimento, Combustivel combustivel, double valorAbastecido, double distanciaPercorrida) {
        super("carro", modelo, placa, capacidade, combustivelInicial, dataAbastecimento, combustivel, valorAbastecido);
        this.distanciaPercorrida = distanciaPercorrida;
    }

    @Override
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    @Override
    public double getConsumo() {
        return distanciaPercorrida / getCombustivelInicial();
    }
}

class Caminhao extends Veiculo {
```

```
    private double distanciaPercorrida;

    public Caminhao(String modelo, String placa, double capacidade, double combustivelInicial,
Date dataAbastecimento, Combustivel combustivel, double valorAbastecido, double
distanciaPercorrida) {
        super("caminhao", modelo, placa, capacidade, combustivelInicial, dataAbastecimento,
combustivel, valorAbastecido);
        this.distanciaPercorrida = distanciaPercorrida;
    }

    @Override
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    @Override
    public double getConsumo() {
        return distanciaPercorrida / getCombustivelInicial();
    }
}

class Moto extends Veiculo {
    private double distanciaPercorrida;

    public Moto(String modelo, String placa, double capacidade, double combustivelInicial, Date
dataAbastecimento, Combustivel combustivel, double valorAbastecido, double
distanciaPercorrida) {
        super("moto", modelo, placa, capacidade, combustivelInicial, dataAbastecimento,
combustivel, valorAbastecido);
        this.distanciaPercorrida = distanciaPercorrida;
    }

    @Override
```

```
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    @Override
    public double getConsumo() {
        return distanciaPercorrida / getCombustivelInicial();
    }
}
```

Herança:

A herança está presente nas classes `Carro`, `Caminhao` e `Moto`, que estendem a classe abstrata `Veiculo`. Isso significa que as classes derivadas herdam os atributos e métodos da classe base.

```java
class Carro extends Veiculo {
    private double distanciaPercorrida;

    // ...
}

class Caminhao extends Veiculo {
    private double distanciaPercorrida;

    // ...
}

class Moto extends Veiculo {
    private double distanciaPercorrida;
```

```
    // ...
}
```

Polimorfismo:

O polimorfismo é evidente na sobrescrita dos métodos `getDistanciaPercorrida()` e `getConsumo()` nas classes derivadas `Carro`, `Caminhao` e `Moto`. Cada uma dessas classes fornece uma implementação específica desses métodos.

```java
class Carro extends Veiculo {
    private double distanciaPercorrida;

    @Override
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    @Override
    public double getConsumo() {
        return distanciaPercorrida / getCombustivelInicial();
    }

    // ...
}

class Caminhao extends Veiculo {
    private double distanciaPercorrida;
```

```
    @Override
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    @Override
    public double getConsumo() {
        return distanciaPercorrida / getCombustivelInicial();
    }

    // ...
}

class Moto extends Veiculo {
    private double distanciaPercorrida;

    @Override
    public double getDistanciaPercorrida() {
        return distanciaPercorrida;
    }

    @Override
    public double getConsumo() {
        return distanciaPercorrida / getCombustivelInicial();
    }

    // ...
}
```