FATEC ZONA SUL

Dom Paulo Evaristo Arns

28 de Novembro 2024

Alexsandro de Jesus Abreu

Luiza Silva Soares Calixto

1. Singleton (Criacional)

Definição:

O padrão Singleton assegura que uma classe tenha apenas uma única instância em todo o ciclo de vida da aplicação e fornece um ponto global de acesso a essa instância.

Aplicação no exemplo:

- A classe `EstoqueTransporte` utiliza o Singleton para garantir que o gerenciamento do estoque seja centralizado.
- Apenas uma instância de estoque é criada, independentemente de quantas vezes seja acessada.

Por que usar?

- Ideal quando é necessário controlar recursos compartilhados, como o estoque de uma transportadora.

Pacote: singleton

Classe : SingletonFX

```
package singleton;


import javafx.application.Application;
```

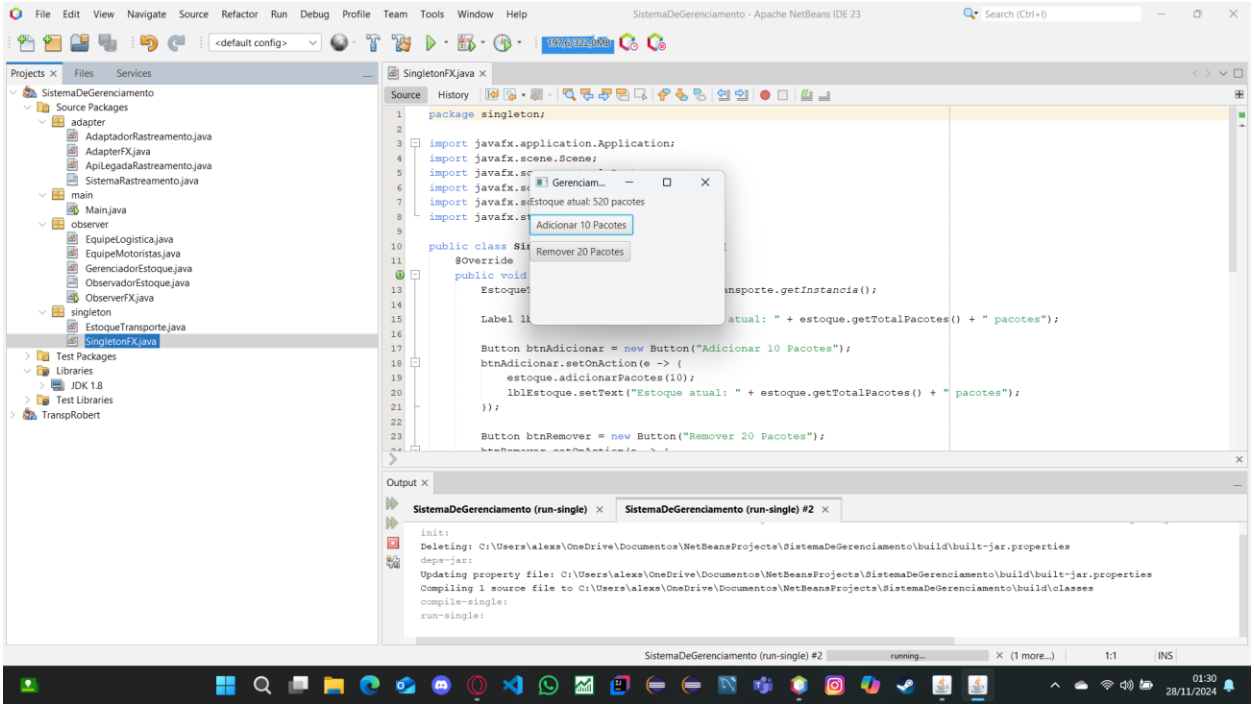
```
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class SingletonFX extends Application {
  @Override
  public void start(Stage palco) {
    EstoqueTransporte estoque = EstoqueTransporte.getInstancia();

    Label lblEstoque = new Label("Estoque atual: " + estoque.getTotalPacotes() + "
pacotes");

    Button btnAdicionar = new Button("Adicionar 10 Pacotes");
    btnAdicionar.setOnAction(e -> {
      estoque.adicionarPacotes(10);
      lblEstoque.setText("Estoque atual: " + estoque.getTotalPacotes() + " pacotes");
    });

    Button btnRemover = new Button("Remover 20 Pacotes");
    btnRemover.setOnAction(e -> {
      estoque.removerPacotes(20);
      lblEstoque.setText("Estoque atual: " + estoque.getTotalPacotes() + " pacotes");
    });
```

```
        VBox layout = new VBox(10, lblEstoque, btnAdicionar, btnRemover);
        Scene cena = new Scene(layout, 300, 200);

        palco.setTitle("Gerenciamento de Estoque - Singleton");
        palco.setScene(cena);
        palco.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Classe : EstoqueTransporte

```
package singleton;

public class EstoqueTransporte {
    private static EstoqueTransporte instancia;
    private int totalPacotes;

    private EstoqueTransporte() {
        totalPacotes = 500; // Estoque inicial
    }

    public static EstoqueTransporte getInstancia() {
```

```
        if (instancia == null) {
            instancia = new EstoqueTransporte();
        }
        return instancia;
    }

    public int getTotalPacotes() {
        return totalPacotes;
    }

    public void adicionarPacotes(int quantidade) {
        totalPacotes += quantidade;
    }

    public void removerPacotes(int quantidade) {
        if (quantidade <= totalPacotes) {
            totalPacotes -= quantidade;
        } else {
            System.out.println("Estoque insuficiente!");
        }
    }
}
```



Print do Singleton

2. Adapter (Estrutural)

Definição:
O padrão Adapter converte a interface de uma classe em outra interface que o cliente espera. Ele é usado para integrar classes incompatíveis sem alterar seus códigos.

Aplicação no exemplo:

- A classe `AdaptadorRastreamento` permite que uma API legada (`ApiLegadaRastreamento`) seja usada com o sistema moderno de rastreamento (`SistemaRastreamento`).
- Adapta a funcionalidade da API antiga para atender aos requisitos da nova interface.

Por que usar?

- Necessário em sistemas onde o legado precisa ser mantido, mas a funcionalidade deve ser atualizada para atender novas demandas.

Pacote: adapter

Classe : AdaptadorRastreamento

```
package adapter;

public class AdaptadorRastreamento implements SistemaRastreamento {
   private ApiLegadaRastreamento apiLegada;

   public AdaptadorRastreamento(ApiLegadaRastreamento apiLegada) {
      this.apiLegada = apiLegada;
   }

   @Override
   public void rastrearPacote(String idPacote) {
      apiLegada.localizar(idPacote);
   }
}
```

Classe : adapter

```
package adapter;

import javafx.application.Application;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class AdapterFX extends Application {
  @Override
  public void start(Stage palco) {
    ApiLegadaRastreamento apiLegada = new ApiLegadaRastreamento();
    SistemaRastreamento adaptador = new AdaptadorRastreamento(apiLegada);

    TextField txtPacote = new TextField();
    txtPacote.setPromptText("Digite o ID do pacote");

    Button btnRastrear = new Button("Rastrear Pacote");
    btnRastrear.setOnAction(e -> {
      String idPacote = txtPacote.getText();
      adaptador.rastrearPacote(idPacote);
    });

    VBox layout = new VBox(10, txtPacote, btnRastrear);
    Scene cena = new Scene(layout, 300, 150);

    palco.setTitle("Sistema de Rastreamento - Adapter");
    palco.setScene(cena);
    palco.show();
```

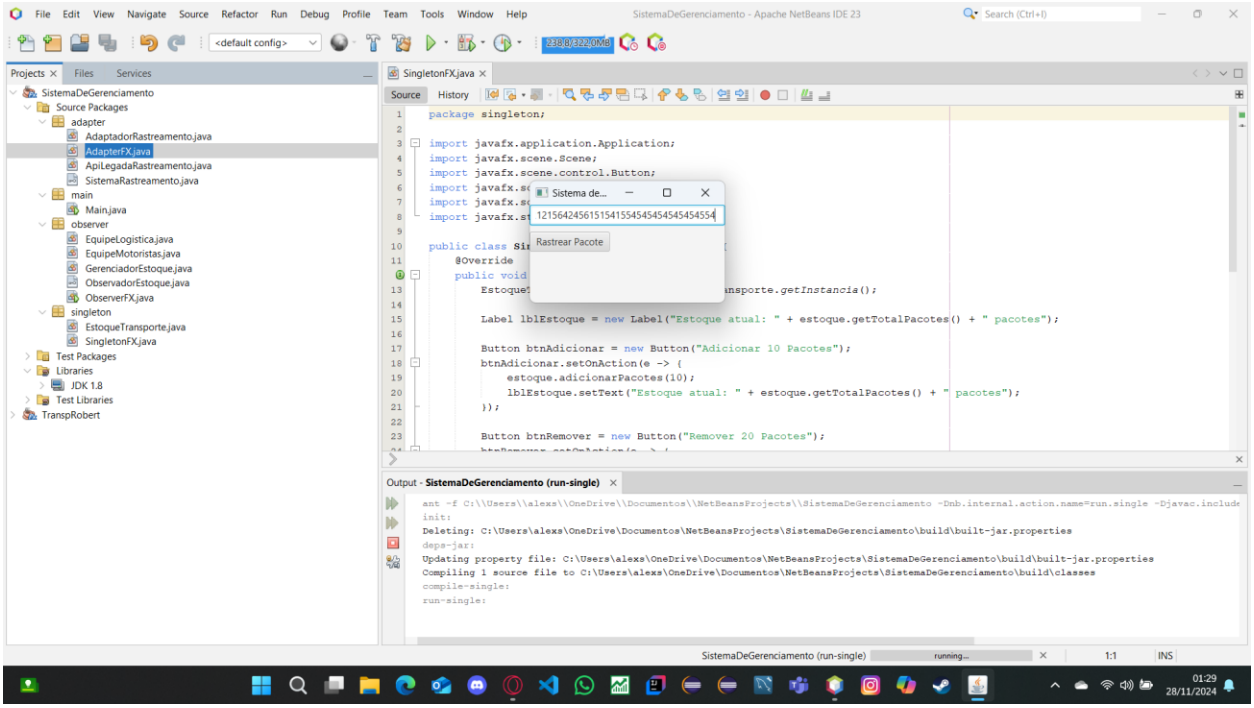
```
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Classe : ApiLegadaRastreamento

```
package adapter;

public class ApiLegadaRastreamento {
    public void localizar(String id) {
        System.out.println("Pacote " + id + " localizado pela API legada.");
    }
}
```

Classe : SistemaRastreamento

```
package adapter;

public interface SistemaRastreamento {
    void rastrearPacote(String idPacote);
}
```



Print do **Adapter**

3. Observer (Comportamental)

Definição:

O padrão Observer estabelece uma dependência de "um para muitos", onde uma mudança no estado de um objeto (o sujeito) notifica automaticamente todos os seus observadores.

Aplicação no exemplo:

- A classe `GerenciadorEstoque` notifica os setores (`EquipeLogistica`, `EquipeMotoristas`) sobre mudanças no estoque.
- Os observadores respondem automaticamente às alterações, garantindo que todos sejam informados.

Por que usar?

- Ideal para sistemas que exigem notificações em tempo real, como mudanças de estoque em uma transportadora.

Pacote: observer

Classe : EquipeLogistica

```
package observer;

public class EquipeLogistica implements ObservadorEstoque {
   @Override
   public void atualizar(String mensagem) {
      System.out.println("Equipe Logística: " + mensagem);
   }
}
```

Classe : EquipeMotoristas

```
package observer;

public class EquipeMotoristas implements ObservadorEstoque {
   @Override
   public void atualizar(String mensagem) {
      System.out.println("Equipe Motoristas: " + mensagem);
   }
}
```

Classe : GerenciadorEstoque

```
package observer;
```

```
import java.util.ArrayList;
import java.util.List;

public class GerenciadorEstoque {
    private List<ObservadorEstoque> observadores = new ArrayList<>();

    public void adicionarObservador(ObservadorEstoque observador) {
        observadores.add(observador);
    }

    public void notificarObservadores(String mensagem) {
        for (ObservadorEstoque obs : observadores) {
            obs.atualizar(mensagem);
        }
    }

    public void alterarEstoque(String operacao, int quantidade) {
        String mensagem = "Operação: " + operacao + ", Quantidade: " + quantidade;
        notificarObservadores(mensagem);
    }
}
```

Classe : ObservadorEstoque

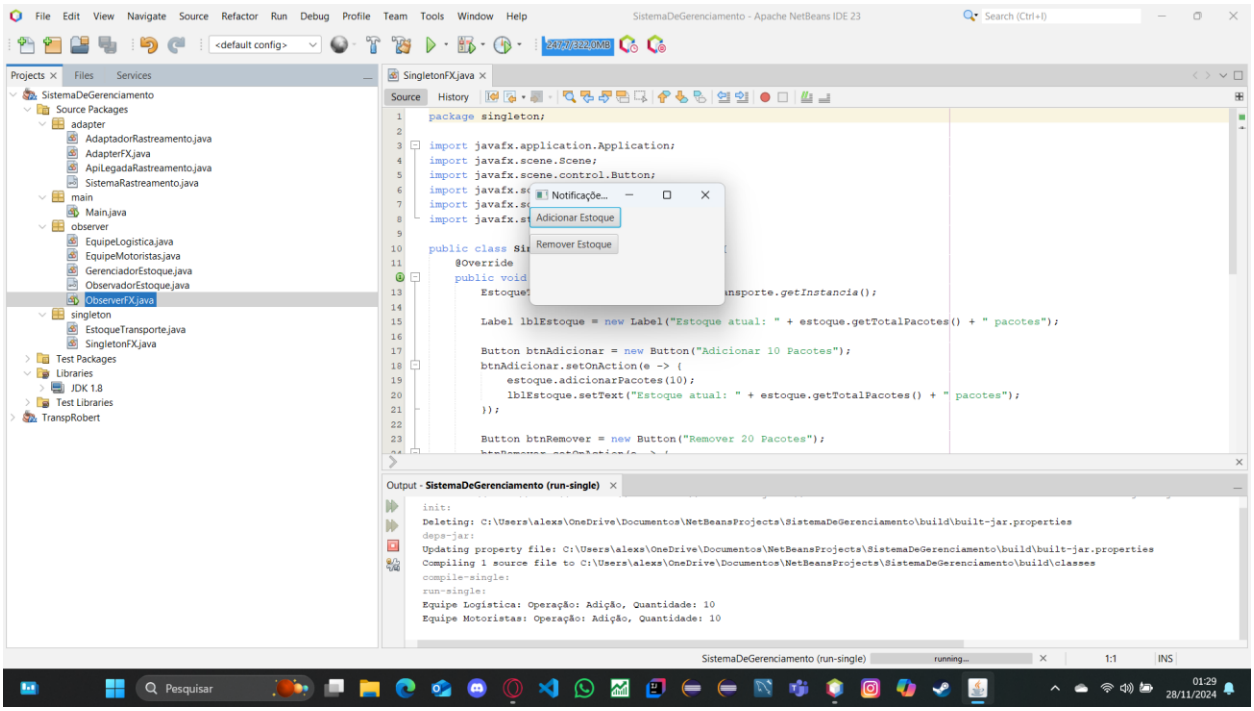
```
package observer;

public interface ObservadorEstoque {
```

```
    void atualizar(String mensagem);
}
```

Classe : ObserverFX

```
package observer;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ObserverFX extends Application {
    @Override
    public void start(Stage palco) {
        GerenciadorEstoque gerenciador = new GerenciadorEstoque();
        gerenciador.adicionarObservador(new EquipeLogistica());
        gerenciador.adicionarObservador(new EquipeMotoristas());

        Button btnAdicionar = new Button("Adicionar Estoque");
        btnAdicionar.setOnAction(e -> gerenciador.alterarEstoque("Adição", 10));

        Button btnRemover = new Button("Remover Estoque");
        btnRemover.setOnAction(e -> gerenciador.alterarEstoque("Remoção", 5));

        VBox layout = new VBox(10, btnAdicionar, btnRemover);
```

```
        Scene cena = new Scene(layout, 300, 150);


        palco.setTitle("Notificações de Estoque - Observer");
        palco.setScene(cena);
        palco.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```



Print do Observer

Classe `Main`

A classe `Main` é o ponto de entrada principal do programa, responsável por integrar e demonstrar o funcionamento dos três padrões de design escolhidos: **Singleton**, **Adapter** e

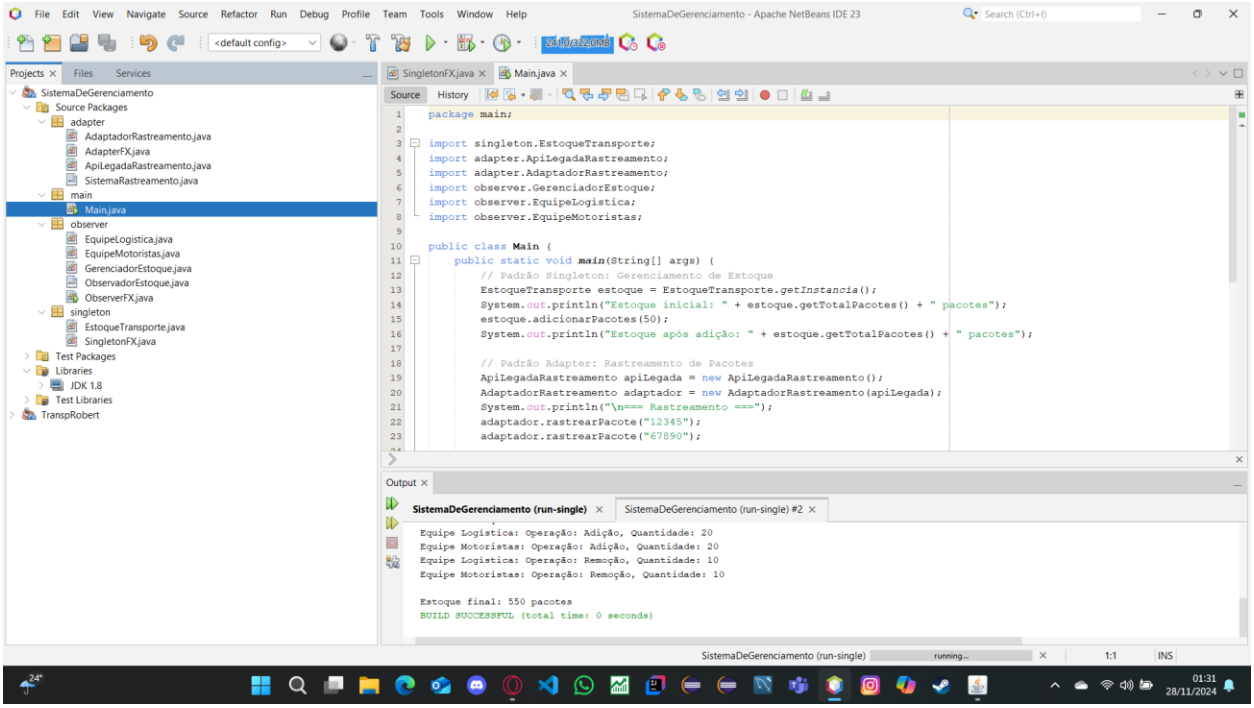
Observer. Cada padrão é aplicado de forma prática para um cenário de **gerenciamento de estoque de uma transportadora**.

Pacote: Main

Classe : Main

```
package main;


import singleton.EstoqueTransporte;
import adapter.ApiLegadaRastreamento;
import adapter.AdaptadorRastreamento;
import observer.GerenciadorEstoque;
import observer.EquipeLogistica;
import observer.EquipeMotoristas;


public class Main {
    public static void main(String[] args) {
        // Padrão Singleton: Gerenciamento de Estoque
        EstoqueTransporte estoque = EstoqueTransporte.getInstancia();
        System.out.println("Estoque inicial: " + estoque.getTotalPacotes() + " pacotes");
        estoque.adicionarPacotes(50);
        System.out.println("Estoque após adição: " + estoque.getTotalPacotes() + " pacotes");
```

```
        // Padrão Adapter: Rastreamento de Pacotes
        ApiLegadaRastreamento apiLegada = new ApiLegadaRastreamento();
        AdaptadorRastreamento adaptador = new AdaptadorRastreamento(apiLegada);
        System.out.println("\n=== Rastreamento ===");
        adaptador.rastrearPacote("12345");
        adaptador.rastrearPacote("67890");

        // Padrão Observer: Notificações de Estoque
        GerenciadorEstoque gerenciador = new GerenciadorEstoque();
        gerenciador.adicionarObservador(new EquipeLogistica());
        gerenciador.adicionarObservador(new EquipeMotoristas());

        System.out.println("\n=== Notificações ===");
        gerenciador.alterarEstoque("Adição", 20);
        gerenciador.alterarEstoque("Remoção", 10);

        // Conclusão
        System.out.println("\nEstoque final: " + estoque.getTotalPacotes() + " pacotes");
    }
}
```



Print do `Main`