Roteiro 08

Neste roteiro iremos explorar o padrão Fatory. Trataremos mais especificamente da chamada **Simple Fatory** ou Concrete Factory . O padrão Fatory tem variações conhecidas por **Fatory Method** e **Abstract Factory**, mas que não serão exploradas neste roteiro.

O **Simple Fatory** ou Fábrica Simples – Permite desacoplar a criação de objetos dos seus códigos clientes. Os padrões são classificados por finalidade de: criação, estrutural ou comportamental. Ou seja, trata-se um padrão de projeto de criação que fornece uma interface para criação de objetos.

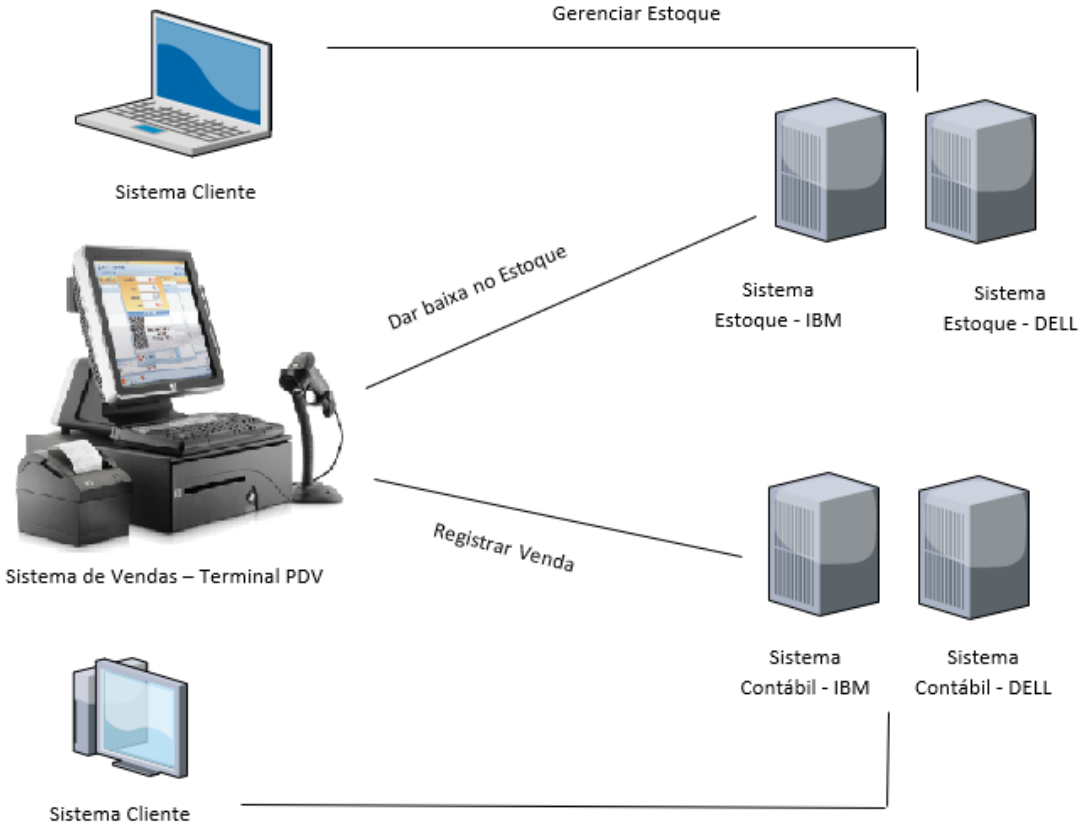
Falando de forma genérica, uma Fabrica Simples geralmente não tem subclasses, mas se adicionarmos uma subclasse a ela, estaremos bem próximos do padrão Fatory Method.

Cenário:

Imagine que temos 2 servidores remotos e as máquinas cliente com sistemas que precisam se integrar a um sistema de estoque através destes servidores. Observem no diagrama a seguir que temos sistemas de estoque diferentes rodando em cada servidor (Um da IBM e outro da DELL). Vejam que temos diferentes perfis de acesso dos clientes para o sistema de estoque (Um perfil gerencial e outro operacional em um terminal PDV).

Temos também 2 sistemas de contabilidade rodando em 2 servidores também (IBM e DELL). De forma semelhante temos um cliente com perfil gerencial e outro operacional.

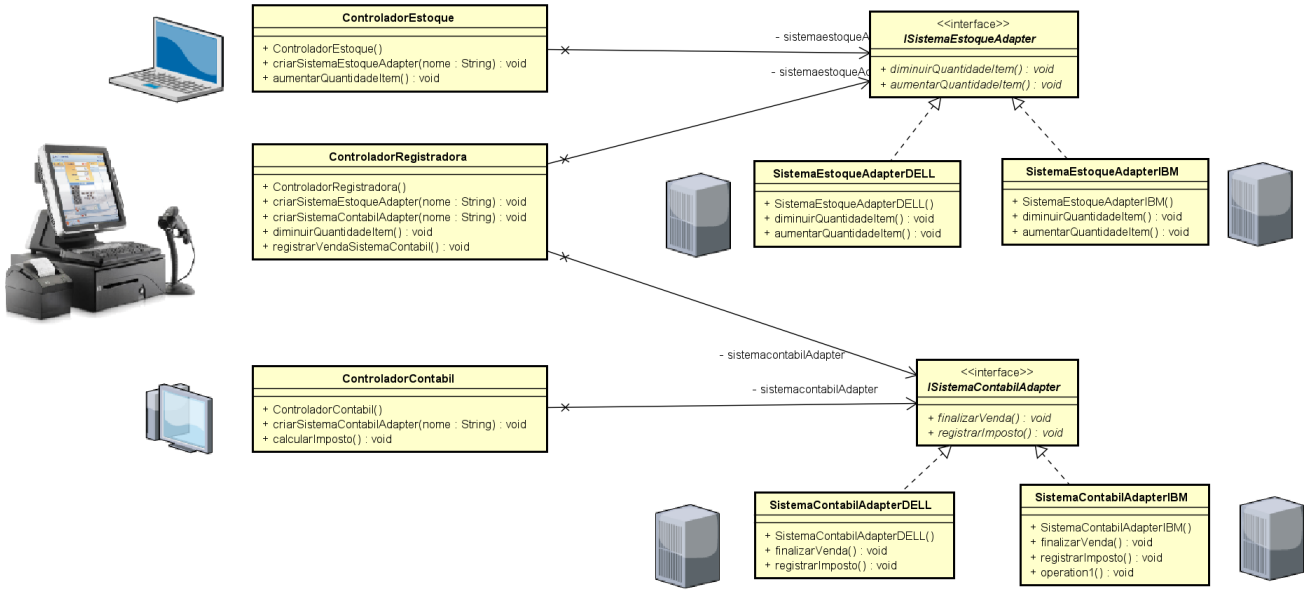
O objetivo é desenvolver uma modelagem que seja possível determinar quais objetos deverão ser criados no startup da aplicação. Ou seja, quando a aplicação na máquina cliente for iniciada devemos ter de forma relativamente fácil de se conectar com o sistema através de um dos servidores (IBM ou DELL).



Para desenvolver o roteiro para este cenário iremos usar uma estratégia um pouco diferente. Na maior parte dos roteiros anteriores optamos por evoluir o projeto a partir do código até chegar em um modelo interessante de padrão de projeto. Desta vez, já iremos partir de uma modelagem e em seguida iremos para o código.

No modelo temos 3 clientes na solução. Um cliente desktop responsável por gerenciar o estoque. Um cliente representado pelo terminal de vendas que precisa acessar o sistema de estoque e possivelmente irá gerar algum registro para a contabilidade. E um terceiro cliente responsável por gerenciar a contabilidade.

Observando o diagrama a seguir percebam que o primeiro objeto a ser instanciado pelos "Sistemas Cliente" deve ser um objeto chamado controlador (**ControladorEstoque**, **ControladorRegistradora**, **ControladorContabil**). Este controlador deverá definir o fluxo de execução da aplicação.



Os "Sistemas Clientes" irão se integrar aos Sistemas de Estoque e Contábil em diferentes servidores a depender a situação. Para esta integração usaremos também o padrão de projeto Adapter, já conhecido em roteiros anteriores. E para cada sistema temos no modelo uma hierarquia de classes específica usando o padrão Adapter.

- **Sistema de Estoque**:
  Interface: ISistemaEstoqueAdapter
  Classes Concretas: SistemaEstoqueAdapterDELL, SistemaEstoqueAdapterIBM
- **Sistema de Contabilidade:**
  Interface : ISistemaContabilAdapter
  Classes Concretas: SistemaContabilAdapterDELL, SistemaContabilIBM

Simulando que os sistemas de Estoque e de Contabilidade são sistemas legados de diferentes fornecedores, deixamos disponível os projetos de cada sistema para promovermos a integração entre os sistemas. São eles: **Roteiro8_SistemaEstoque, Roteiro8_SistemaContabil** (Disponíveis na pasta de trabalho)

Início do projeto – Pacote : roteiro8.parte1

1 – Dentro do projeto criar um pacote chamado **roteiro8.parte1**

2 – Inicialmente crie as interfaces **ISistemaEstoqueAdapter** e **ISistemaContabilAdapter** dentro do pacote **roteiro8.parte1.**

```
package roteiro8.parte1;

public interface ISistemaContabilAdapter {
   void finalizarVenda();
   void registrarImposto();
}
```

```
package roteiro8.parte2;

public interface ISistemaEstoqueAdapter {
  public void diminuirQuantidadeItem();
  public void aumentarQuantidadeItem();
}
```

3 – Temos agora a interface de cada hierarquia referente aos sistemas que iremos integrar. Na parte 1 deste roteiro vamos nos concentrar em desenvolver todo o fluxo para o Sistema de Contabilidade.

Vejam abaixo que temos as classes concretas do Sistema Contábil para o servidor DELL e IBM. Elas implementam a interface **ISistemaContabilAdapter** e fazem referência ao sistema legado usando **import domínio.SistemaContabil**. Este sistema está disponível na pasta de trabalho (Roteiro8_SistemaContabil).

Observem que estamos aplicando o padrão Adapter semelhante ao que fizemos do Roteiro 04. Cada método desta classe chama um método correspondente do outro sistema por meio de delegação.

```
package roteiro8.parte1;

import dominio.SistemaContabil;

public class SistemaContabilAdapterDELL implements
ISistemaContabilAdapter {

   private SistemaContabil sistemacontabil;

   public SistemaContabilAdapterDELL() {
      this.sistemacontabil = new SistemaContabil("DELL");
   }

   @Override
   public void finalizarVenda() {
      this.sistemacontabil.registrarVenda();
   }

   @Override
   public void registrarImposto() {
      this.sistemacontabil.calcularImposto();
   }
}
```

```
package roteiro8.parte1;

import dominio.SistemaContabil;

public class SistemaContabilAdapterIBM implements
ISistemaContabilAdapter{

   private SistemaContabil sistemacontabil;

   public SistemaContabilAdapterIBM() {
      this.sistemacontabil = new SistemaContabil("IBM");
   }

   @Override
   public void finalizarVenda() {
      this.sistemacontabil.registrarVenda();
   }

   @Override
   public void registrarImposto() {
      this.sistemacontabil.calcularImposto();
   }
}
```

4 – Criaremos agora a classe **ControladorContabil** que será responsável por controlar quem será o objeto a ser criado. O sistema cliente irá acionar o controlador e ele direcionar para o Sistema Contábil da DELL ou da IBM.

```
package roteiro8.parte1;

public class ControladorContabil {

   private ISistemaContabilAdapter sistemacontabilAdapter;

   public ControladorContabil() {
      System.out.println("Controlador de Sistema Contabil Criado");
   }

   public void criarSistemaContabilAdapter(String nome){
      if (nome.equals("IBM"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterIBM();
      else if (nome.equals("DELL"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterDELL();
   }

   public void calcularImposto(){
      this.sistemacontabilAdapter.registrarImposto();
   }
}
```

5 – Crie agora a classe TesteControlador para validarmos o fluxo dos sistemas.

```
package roteiro8.parte1;

public class TesteControlador {

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

      public static void testeControladorContabil(){
      System.out.println("Criando o Controlador Contabil");
      ControladorContabil controladorcontabil = new ControladorContabil();

      System.out.println("Teste de Integração do Controlador Contabil - Sistema Contabil DELL");
      controladorcontabil.criarSistemaContabilAdapter("DELL");
      controladorcontabil.calcularImposto();

      System.out.println("Teste de Integração do Controlador Contabil - Sistema Contabil IBM");
      controladorcontabil.criarSistemaContabilAdapter("IBM");
      controladorcontabil.calcularImposto();
   }
}
```

Pacote : roteiro8.parte2

Na parte 1 fizemos o fluxo para o Sistema de Contabilidade. Na parte 2 vamos aplicar a mesma lógica e aplicar ao Sistema de Estoque.

Vamos ao passo a passo deste pacote

1 – No mesmo projeto crie o pacote roteiro8.parte2

2 – Copie todas as classes criadas na parte1 para o novo pacote.

3 – Na parte 1 já criamos a interface ISistemaEstoqueAdapter. Agora devemos criar as classes concretas que implementam esta interface: **SistemaEstoqueAdapterDELL** e **SistemaEstoqueAdapterIBM**. Semelhante ao que foi feito na parte1 com o Sistema Contábil.

Lembrando que estas classes devem fazer referência ao Sistema de Estoque legado disponível na pasta de trabalho (Roteiro8_SistemaEstoque), usando **import servico.SistemaEstoque**.

Segue abaixo os métodos correspondentes para aplicar o padrão Adapter:

- public void diminuirQuantidadeItem() : Equivale ao método removerItemEstoque() no Sistema de Estoque.

- public void aumentarQuantidadeItem() : Equivale ao método adicionarItemEstoque no Sistema de Estoque.

4 – Seguindo a mesma lógica agora crie a classe **ControladorEstoque**. Lembre-se de implementar os seguintes atributos e métodos.

Atributo:

private ISistemaEstoqueAdapter sistemaestoqueAdapter; - Para se relacionar com o Sistema de Estoque.

Construtor:

public ControladorEstoque()

Método para Criação dos Objetos:

public void criarSistemaEstoqueAdapter(String nome)

Métodos Específicos:

public void aumentarQuantidadeItem() - Neste método faça a implementação da seguinte forma

```
public void aumentarQuantidadeItem(){
    this.sistemaestoqueAdapter.aumentarQuantidadeItem();
  }
```

5 – Acrescente um novo trecho na classe **TesteControlador** e faça os devidos testes.

```
package roteiro8.parte2;

public class TesteControlador {

   public static void main(String[] args) {
      testeControladorContabil();
      testeControladorEstoque();
   }

   public static void testeControladorContabil(){
      System.out.println("Criando o Controlador Contabil");
      ControladorContabil controladorcontabil = new ControladorContabil();

      System.out.println("Teste de Integração do Controlador Contabil - Sistema Contabil DELL");
      controladorcontabil.criarSistemaContabilAdapter("DELL");
      controladorcontabil.calcularImposto();

      System.out.println("Teste de Integração do Controlador Contabil - Sistema Contabil IBM");
      controladorcontabil.criarSistemaContabilAdapter("IBM");
      controladorcontabil.calcularImposto();
   }

   public static void testeControladorEstoque(){
      System.out.println("Criando o Controlador de Estoque");
      ControladorEstoque controladorestoque = new ControladorEstoque();

      System.out.println("Teste de Integração do Controlador de Estoque - Sistema de Estoque DELL");
      controladorestoque.criarSistemaEstoqueAdapter("DELL");
      controladorestoque.aumentarQuantidadeItem();

      System.out.println("Teste de Integração do Controlador de Estoque - Sistema de Estoque IBM");
      controladorestoque.criarSistemaEstoqueAdapter("IBM");
      controladorestoque.aumentarQuantidadeItem();
   }

}
```

Pacote : roteiro8.parte3

Nas partes 1 e 2 fizemos o fluxo para o Sistema de Contabilidade e o Sistema de Estoque. Mas, o terminal de vendas PDV com o Sistema de Vendas que é mais um "Sistema Cliente". E neste caso específico ele precisa se integrar tanto ao Estoque quanto a Contabilidade. Para completar este fluxo precisamos apenas criar o controlador para a registradora. Termos então 3 Controladores : **ControladorEstoque**, **ControladorContabil e ControladorRegistradora**.

Vamos ao passo a passo deste pacote

1 – No mesmo projeto crie o pacote roteiro8.parte3

2 – Copie todas as classes criadas na parte2 para o novo pacote.

3 – Seguindo a mesma lógica agora crie a classe **ControladorRegistradora**. Lembre-se de implementar os seguintes atributos e métodos. Neste caso deste controlador termos 2 atributos. Um para se relacionar com o sistema de estoque e outro para se relacionar com o sistema de contabilidade.

Atributo:

private ISistemaContabilAdapter sistemacontabilAdapter; - Para se relacionar com o Sistema Contábil

private ISistemaEstoqueAdapter sistemaestoqueAdapter; - Para se relacionar com o Sistema de Estoque

Construtor:

public ControladorRegistradora()

Método para Criação dos Objetos:

public void criarSistemaContabilAdapter(String nome)

public void criarSistemaEstoqueAdapter(String nome)

Métodos Específicos:

```
public void diminuirQuantidadeItem(){
    this.sistemaestoqueAdapter.diminuirQuantidadeItem();
  }
```

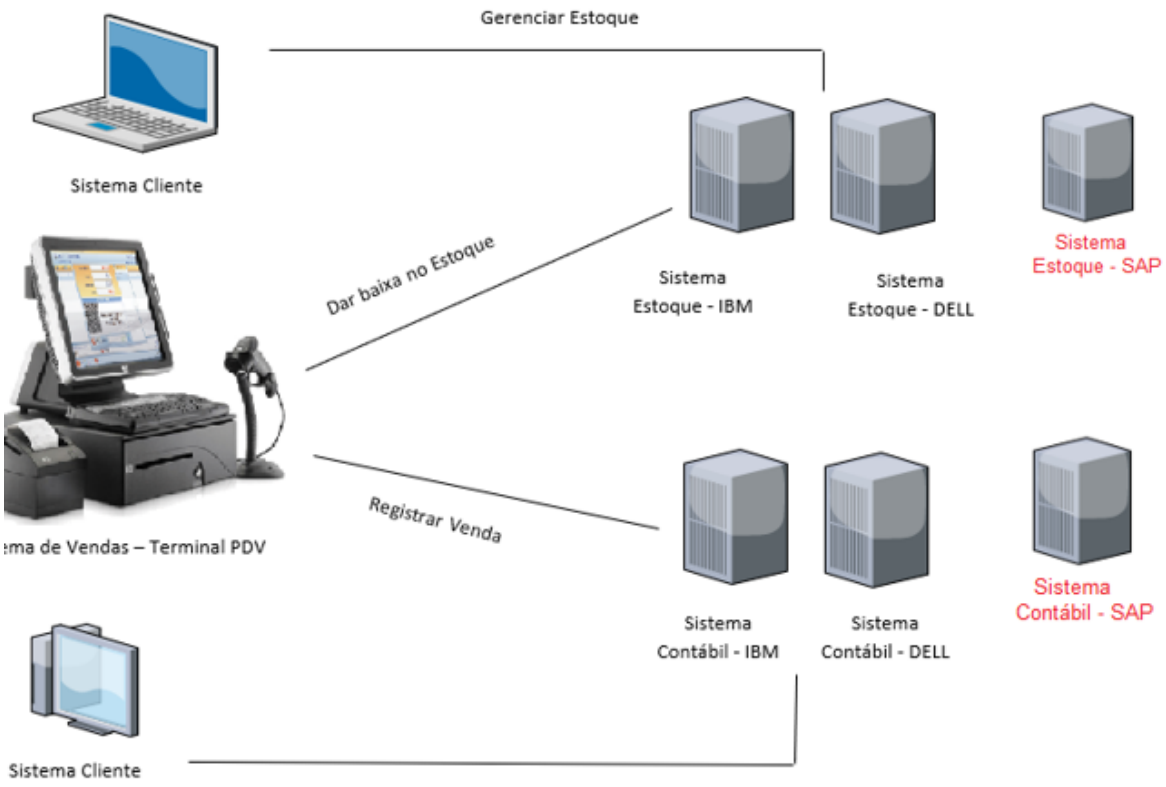
```
public void registrarVendaSistemaContabil(){
    this.sistemacontabilAdapter.finalizarVenda();
  }
```

4 – Acrescente um novo trecho na classe **TesteControlador** e faça os devidos testes. Segue abaixo um novo método a ser adicionado referente ao teste da ControladoraRegistradora

```
public static void testeControladoraRegistradora(){
  System.out.println("Criando o Controlador da Registradora");
  ControladorRegistradora controladorregistradora = new ControladorRegistradora();

  System.out.println("Teste de Integração do Controlador da Registradora - Sistema de Estoque DELL");
  controladorregistradora.criarSistemaEstoqueAdapter("DELL");
  controladorregistradora.diminuirQuantidadeItem();

  System.out.println("Teste de Integração do Controlador da Registradora - Sistema de Estoque IBM");
  controladorregistradora.criarSistemaEstoqueAdapter("IBM");
  controladorregistradora.diminuirQuantidadeItem();

  System.out.println("Teste de Integração do Controlador da Registradora - Sistema Contabil DELL");
  controladorregistradora.criarSistemaContabilAdapter("DELL");
  controladorregistradora.registrarVendaSistemaContabil();

  System.out.println("Teste de Integração do Controlador da Registradora - Sistema Contabil IBM");
  controladorregistradora.criarSistemaContabilAdapter("IBM");
  controladorregistradora.registrarVendaSistemaContabil();
}
```

Pacote : roteiro8.parte4

Como todo projeto evolui, imagine que agora teremos o Servidor SAP como um novo servidor alternativo a nossa solução. O diagrama abaixo ganhou outro servidor



Vamos ao passo a passo deste pacote

1 – No mesmo projeto crie o pacote roteiro8.parte4

2 – Copie todas as classes criadas na parte3 para o novo pacote.

3 – Implemente a classe Sistema **SistemaContabilAdapterSAP** de forma semelhante as outras classes irmãs.

4 - Implemente a classe Sistema **SistemaEstoqueAdapterSAP** de forma semelhante as outras classes irmãs.

5 – Registre aqui os impactos que os itens 3 e 4 terão nas classes controladoras e em seguida faça as implementações.

6 – Faça as intervenções necessárias na classe **TesteControlador** e proceda com os testes.

7 – Já consegue visualizar as desvantagens desta modelagem? Registre aqui a sua percepção.

Pacote : roteiro8.parte5

Tentaremos nesta parte fazer uma pequena melhoria da hierarquia do padrão Adapter, pois percebemos um grande numero de repetição de código nas classes concretas.

Uma possível solução seria substituir as interfaces **ISistemaContabilAdapter** e **ISistemaEstoqueAdapter** por classes abstratas e os trechos de código que se repetem nas classes concretas viriam para a classe abstrata.

Vamos ao passo a passo deste pacote

1 – No mesmo projeto crie o pacote roteiro8.parte5

2 – Copie todas as classes criadas na parte4 para o novo pacote.

3 – Crie a classe abstrata **SistemaContabilAdapter** conforme segue abaixo e exclua a interface **ISistemaContabilAdapter**. Observe os seguintes pontos:

- O import domínio.SistemaContabil precisa ser feito na classe abstrata também.

- Os métodos finalizarVenda() e registrarImposto() saem das classes concretas e vão para a classe abstrata.

- As classes concretas agora herdam (extends) da classe abstrata

Segue abaixo o resultado final da classe abstrata para o Sistema Contábil e uma das classes concretas (DELL).

```
package roteiro8.parte5;

import dominio.SistemaContabil;

public abstract class SistemaContabilAdapter {

   protected SistemaContabil sistemacontabil;

   public void finalizarVenda() {
      this.sistemacontabil.registrarVenda();
   }

   public void registrarImposto() {
      this.sistemacontabil.calcularImposto();
   }
}
```

```
package roteiro8.parte5;

import dominio.SistemaContabil;

public class SistemaContabilAdapterDELL extends SistemaContabilAdapter {

   public SistemaContabilAdapterDELL() {
      this.sistemacontabil = new SistemaContabil("DELL");
   }
}
```

O mesmo deve ser feito nas classes concretas IBM e SAP.

4 – Crie a classe abstrata **SistemaEstoqueAdapter** e exclua a interface **ISistemaEstoqueAdapter**. Proceda com a mesma lógica aplicada no item 3.

Pacote : roteiro8.parte6

Faremos agora uma nova melhoria na modelagem no que diz respeito aos controladores, pois percebemos que ao introduzirmos o servidor SAP tivemos além da repetição de código, impactos significativos com intervenções nos 3 controladores (**ControladorEstoque**, **ControladorContabil e ControladorRegistradora**)

Tentaremos fazer uso do padrão Fatory (Fábrica Simples) como foi citado no início deste roteiro.

Vamos ao passo a passo deste pacote

1 – No mesmo projeto crie o pacote roteiro8.parte5

2 – Copie todas as classes criadas na parte5 para o novo pacote.

3 – Crie a classe **AdapterFatory** conforme segue abaixo. Ela irá concentrar os métodos de criação de objetos.

```
package roteiro8.parte6;

public class AdapterFatory {

   private SistemaEstoqueAdapter sistemaestoqueAdapter;
   private SistemaContabilAdapter sistemacontabilAdapter;

   public SistemaContabilAdapter criarSistemaContabilAdapter(String nome){
      if (nome.equals("IBM"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterIBM();
      else if (nome.equals("DELL"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterDELL();
      else if (nome.equals("SAP"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterSAP();

      return this.sistemacontabilAdapter;
   }

   public SistemaEstoqueAdapter criarSistemaEstoqueAdapter(String nome){
      if (nome.equals("IBM"))
         this.sistemaestoqueAdapter = new SistemaEstoqueAdapterIBM();
      else if (nome.equals("DELL"))
         this.sistemaestoqueAdapter = new SistemaEstoqueAdapterIBM();
      else if (nome.equals("SAP"))
         this.sistemaestoqueAdapter = new SistemaEstoqueAdapterIBM();

      return this.sistemaestoqueAdapter;
   }
}
```

4 – Agora faça a refatoração nas classes controladoras (**ControladorContabil**, **ControladorEstoque**, **ControladorRegistradora**). Segue abaixo o exemplo de uma destas classes. Lembre-se de que a classe ControladorRegistradora tem referência tanto para o sistema de estoque quanto para o sistema de contabilidade.

```
package roteiro8.parte6;

public class ControladorContabil {

   private SistemaContabilAdapter sistemacontabilAdapter;
   private AdapterFatory fatory;

   public ControladorContabil() {
      System.out.println("Controlador de Sistema Contabil Criado");
      this.fatory = new AdapterFatory();
   }

   public void criarSistemaContabilAdapter(String nome){
      this.sistemacontabilAdapter = this.fatory.criarSistemaContabilAdapter(nome);
   }

   public void calcularImposto(){
      this.sistemacontabilAdapter.registrarImposto();
   }
}
```

5 – Podemos fazer também com que os objetos criados pela fábrica tenha uma instancia única. Para isso, aplique o padrão singleton na classe AdapterFatory.

```
package roteiro8.parte6;

public class AdapterFatory {

   private SistemaEstoqueAdapter sistemaestoqueAdapter;
   private SistemaContabilAdapter sistemacontabilAdapter;

   private static AdapterFatory instance = new AdapterFatory();

   private AdapterFatory() {

   }

   public static AdapterFatory getInstance(){
      return instance;
   }

   public SistemaContabilAdapter criarSistemaContabilAdapter(String nome){
      if (nome.equals("IBM"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterIBM();
      else if (nome.equals("DELL"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterDELL();
      else if (nome.equals("SAP"))
         this.sistemacontabilAdapter = new SistemaContabilAdapterSAP();

      return this.sistemacontabilAdapter;
   }

   public SistemaEstoqueAdapter criarSistemaEstoqueAdapter(String nome){
      if (nome.equals("IBM"))
         this.sistemaestoqueAdapter = new SistemaEstoqueAdapterIBM();
      else if (nome.equals("DELL"))
         this.sistemaestoqueAdapter = new SistemaEstoqueAdapterIBM();
      else if (nome.equals("SAP"))
         this.sistemaestoqueAdapter = new SistemaEstoqueAdapterIBM();

      return this.sistemaestoqueAdapter;
   }
}
```

6 – Novamente faça a refatoração nas classes controladoras (**ControladorContabil**, **ControladorEstoque**, **ControladorRegistradora**). Segue abaixo o exemplo de uma destas classes.

```
package roteiro8.parte6;

public class ControladorContabil  {

   private SistemaContabilAdapter sistemacontabilAdapter;
   private AdapterFatory fatory;

   public ControladorContabil()  {
      System.out.println("Controlador de Sistema Contabil Criado");
      this.fatory = AdapterFatory.getInstance();
   }

   public void criarSistemaContabilAdapter(String nome){
      this.sistemacontabilAdapter = this.fatory.criarSistemaContabilAdapter(nome);
   }

   public void calcularImposto(){
      this.sistemacontabilAdapter.registrarImposto();
   }
}
```

7 – Utilize uma ferramenta de software qualquer para geração do diagrama de classes para esta etapa do projeto (Sugestão : Astah Community). Obs.: Adicione aqui o diagrama para que seja disponibilizado no teams