



Exemplo de facade pattern



**Certified
Developer**
The Ultimate Tech Degree

DigitalHouse >
Coding School

Exemplo de facade pattern

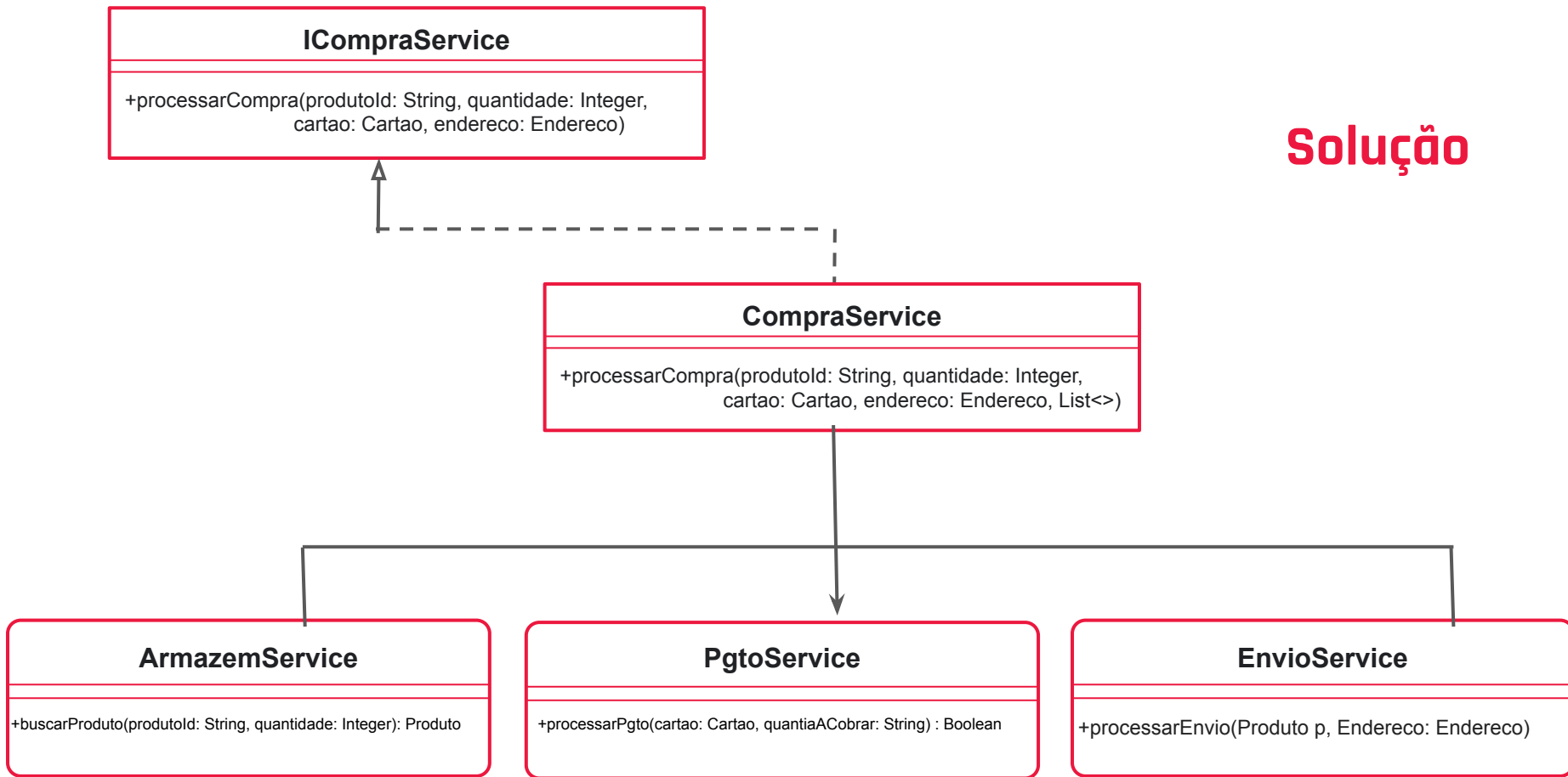
Agora vamos imaginar uma implementação ...

Suponha que tenhamos que projetar um sistema para um e-commerce. Nosso cliente nos pede que na hora de vender o produto, nosso sistema execute uma série de etapas, por exemplo: fazer o pedido do produto no armazém, creditar o pagamento e enviar o pedido.

Vejamos como podemos resolver esse problema aplicando esse padrão.



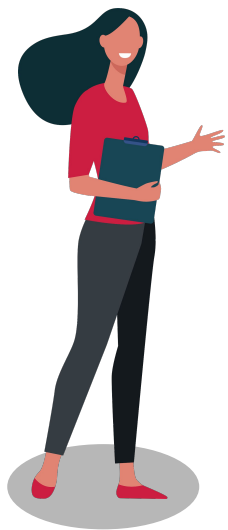
Solução



Implementação das classes do diagrama UML

```
public interface ICompraService {  
  
    public void processarCompra(String produtoId,  
        Integer quantidade, Cartao cartao,  
        Endereco endereco);  
  
}
```

Código da
interface
ICompraService



```
public class CompraService implements ICompraService {  
  
    private ArmazenService armazenService;  
    private PgtoService pgtoService;  
    private EnvioService envioService;  
  
    public CompraService(ArmazenService armazenService, PgtoService  
pgtoService, EnvioService envioService) {  
        this.armazenService = armazenService;  
        this.pgtoService = pgtoService;  
        this.envioService = envioService;  
    }  
}
```

Código da
classe
CompraService
(nossa facade)



```
@Override
public void processarCompra(String produtoId, Integer quantidade, Cartao
cartao, Endereco endereco) {
    Produto produto = armazenService.buscarProduto(produtoId, quantidade);
    if (produto != null) {
        //Se temos o produto, processamos o pagamento
        if (pgtoService.processarPgto(cartao, String.valueOf(produto.getValor() *
quantidade))) {
            envioService.processarEnvio(Arrays.asList(produto), endereco);
        }
    }
}
```

Código da
classe
CompraService
(nossa facade)



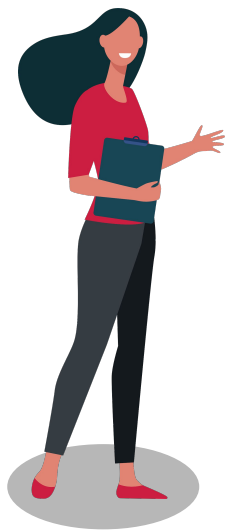
```
public class ArmazenService {  
    private List<Produto> produtos;  
  
    public ArmazenService (List<Produto> produtos)  
    {  
        this.produtos = produtos;  
    }  
}
```

Código da
classe
ArmazenService
(subsistema)



```
public Produto buscarProduto(String produtoId, Integer quantidade) {  
    Produto produto = null;  
    for (Produto p : this.produtos) {  
        if (p.getProdutoId().equals(produtoId) && p.getQuantidade() >=  
quantidade)  
            produto = p;  
        p.setQuantidade(p.getQuantidade() - quantidade); // atualizamos  
a quantidade  
        produto.setQuantidade(quantidade);  
    }  
  
    return produto;  
}  
  
}
```

Código da
classe
ArmazenService
(subsistema)



```
public class PgtoService {  
  
    public Boolean processarPgto(Cartao cartao, String  
    quantiaACobrar){  
        Boolean pgtoRealizado = Boolean.FALSE;  
        if(cartao != null && cartao.getNumerosFrente() != null &&  
        cartao.getCodSeguranca() != null)  
            System.out.println("Processando o pagamento por "+  
            quantiaACobrar);  
            pgtoRealizado = Boolean.TRUE;  
  
        return pgtoRealizado;  
  
    }  
}
```

Código da
classe
PgtoService
(subsistema)



```
public class EnvioService {  
  
    public void processarEnvio (List<Produto> produtos ,  
Endereco endereco) {  
        System.out.println("Enviando produtos a " +  
endereco.getRua() + " " + endereco.getNro() + ", "+  
endereco.getBairro()) ;  
    }  
}
```

Código da
classe
EnvioService
(subsistema)

Simulamos um caso no método main para testá-lo.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Produto produtoUm = new Produto("1",5,1000,"Mouse");  
        Produto produtoDois = new Produto("2",5,3000,"Teclado");  
        Cartao cartao = new Cartao("1111222233334444","012","2025/07/09");  
  
        Endereco endereco = new Endereco("Av Morumbi","1500","14280000","Morumbi","São  
Paulo");  
  
        ICompraService compraService = new CompraService(new  
ArmazenService(Arrays.asList(produtoUm,produtoDois)),new PgtoService(),new  
EnvioService());  
  
        compraService.processarCompra("1",3, cartao, endereco);  
  
    }  
}
```

Conclusão

Através de uma **interface** definimos como o cliente deve se comunicar com nosso sistema; definimos a classe **CompraService** que atuará como uma **fachada (facade)**, recebendo solicitações e se comunicando com os subsistemas para que, em conjunto, a compra seja concluída.



DigitalHouse>
Coding School