INSTITUTO FEDERAL GOIANO - CAMPUS MORRINHOS

5º período do curso de Sistemas para Internet

Disciplina: Padrões de Projeto de Software

Professor: Fernando Barbosa Matos

Aluno: Shayra Kelly Eleotério Silva

Implementação do Padrão de Projeto Adapter

O problema proposto consistia em integrar duas bibliotecas com interfaces incompatíveis em um sistema de gerenciamento de dados para uma empresa de logística. Especificamente, tratava-se de permitir que uma classe legada de armazenamento de veículos ("OldVehicleStorage"), que opera com dados em formato de "String", pudesse ser utilizada através de uma nova interface ("IVehicleStorage"), que trabalha com objetos do tipo "Vehicle".

O padrão Adapter foi escolhido por ser a solução ideal para "traduzir" a interface de uma classe para outra interface esperada pelo cliente, permitindo que classes com interfaces incompatíveis trabalhem juntas.

A solução foi estruturada em torno da criação de uma classe "VehicleStorageAdapter", que atua como o componente central do padrão.

Os principais componentes envolvidos na implementação são:

- **"Vehicle.java" (Model):** Uma classe de entidade simples (POJO) que representa um veículo, com atributos como "id", "model" e "year", além de construtores, getters e setters.
- **"OldVehicleStorage.java" (Adaptee - Classe a ser Adaptada / Model):** Representa a biblioteca legada. Contém o método "storeVehicleData(String data)", que simula o armazenamento dos dados de um veículo recebidos como uma única "String". Esta classe não pôde

ser alterada, conforme o princípio de não modificar código legado sempre que possível.

- ***"IVehicleStorage.java" (Target Interface - Interface Alvo / Model):*** Define a interface que o sistema cliente espera utilizar para o armazenamento de veículos. Possui o método "void saveVehicleData(Vehicle vehicle)".
- ***"VehicleStorageAdapter.java" (Adapter / Model):*** Esta é a classe que implementa o padrão Adapter. Ela implementa a interface "IVehicleStorage" e, internamente, contém uma referência a um objeto "OldVehicleStorage". O adapter é responsável por receber um objeto "Vehicle", convertê-lo para o formato de "String" esperado pela "OldVehicleStorage", e então delegar a chamada para o método "storeVehicleData" da classe legada.

A estrutura de pacotes seguiu uma organização MVC simplificada, com as classes acima localizadas no pacote "model" e a classe de demonstração no pacote "app".

A parte crucial da adaptação ocorre dentro do método "saveVehicleData(Vehicle vehicle)" da classe "VehicleStorageAdapter". Os passos são:

1. O método "saveVehicleData" recebe um objeto "Vehicle" como parâmetro (conforme definido pela interface "IVehicleStorage").
2. Internamente, este método chama um método auxiliar privado, "convertVehicleToString(Vehicle vehicle)".
3. O método "convertVehicleToString" é responsável por transformar os dados do objeto "Vehicle" em uma representação "String". Para esta atividade, foi adotado um formato simples onde os atributos do veículo são concatenados e separados por ponto e vírgula, prefixados por identificadores (ex: "ID:valor;MODEL:valor;YEAR:valor").

java

// Trecho do VehicleStorageAdapter.java

```
private String convertVehicleToString(Vehicle vehicle) {
        StringBuilder sb = new StringBuilder();
        sb.append("ID:").append(vehicle.getId()).append(";");
        sb.append("MODEL:").append(vehicle.getModel()).append(";
");
        sb.append("YEAR:").append(vehicle.getYear());
        return sb.toString();
}
```

O uso de "StringBuilder" foi preferido em relação à concatenação direta de strings por questões de eficiência.

4. A "String" resultante é então passada para o método "storeVehicleData" da instância de "OldVehicleStorage" que o adapter mantém.

Dessa forma, o cliente que utiliza a interface "IVehicleStorage" não tem conhecimento da complexidade da conversão nem da existência da "OldVehicleStorage".

Para demonstrar o funcionamento da solução, foi criada a classe "Main.java" (no pacote "app"). Esta classe executa os seguintes passos:

1. Instancia um objeto "OldVehicleStorage" (simulando a biblioteca legada).
2. Instancia um objeto "VehicleStorageAdapter", passando a instância da biblioteca legada para o seu construtor. Isso estabelece a conexão entre o adapter e o adaptado.
3. Cria um ou mais objetos "Vehicle" com dados de exemplo.
4. Chama o método "saveVehicleData" na instância do "VehicleStorageAdapter", passando os objetos "Vehicle".
5. As saídas no console, geradas tanto pelo "VehicleStorageAdapter" quanto pela "OldVehicleStorage", confirmam que os dados do veículo foram

convertidos e "armazenados" pela biblioteca legada, mesmo a interação inicial tendo ocorrido através da nova interface "IVehicleStorage".

Esta demonstração valida que o adapter cumpre seu papel de intermediário e tradutor entre as interfaces incompatíveis.

Durante a implementação, alguns desafios foram identificados:

- **Formato de Dados da Biblioteca Legada:**
  - Desafio: A "OldVehicleStorage" espera uma "String", mas o formato exato dessa string (ex: CSV, JSON, XML, ou um formato proprietário) não foi especificado em detalhes na descrição inicial do problema. Era preciso definir um formato para a conversão.
  - Solução: Foi adotado um formato de string simples e customizado (chave:valor;chave:valor). Essa lógica de conversão foi encapsulada no método "convertVehicleToString" dentro do "VehicleStorageAdapter". Isso torna o adapter o único local que precisa conhecer os detalhes dessa conversão, facilitando futuras manutenções caso o formato legado mude ou seja mais complexo.
- **Garantia de Não Alteração do Código Legado:**
  - Desafio: Um dos princípios ao lidar com sistemas legados é evitar modificá-los, para não introduzir novos riscos ou quebrar funcionalidades existentes.
  - Solução: O padrão Adapter intrinsecamente resolve este desafio. A classe "OldVehicleStorage" foi utilizada "como está", sem nenhuma alteração em seu código. O "VehicleStorageAdapter" atua como um "envoltório" (wrapper) em torno dela.
- Aplicação dos Conceitos MVC:
  - Desafio: A atividade pedia para realizar a atividade "nos padrões MVC". Embora o Adapter seja um padrão estrutural

e não arquitetural como o MVC, foi necessário pensar em como organizar os componentes de forma coerente.

- Solução: As classes de dados ("Vehicle"), a interface alvo ("IVehicleStorage"), a classe legada ("OldVehicleStorage") e o próprio "VehicleStorageAdapter" foram agrupadas como componentes do "Model", pois lidam com os dados e a lógica de negócio/adaptação. A classe "Main" atuou como um "Controller" muito simples, orquestrando a interação para fins de demonstração, e a "View" foi o console do sistema.

A implementação do padrão de projeto Adapter foi bem-sucedida em atender aos requisitos da atividade. Foi possível integrar a biblioteca legada "OldVehicleStorage" com a nova interface "IVehicleStorage" de forma transparente para o cliente da nova interface. Isso foi alcançado sem modificar o código da biblioteca legada, destacando a principal vantagem do padrão Adapter: promover a colaboração entre classes com interfaces incompatíveis.