FIAP

FÁBRICA DE DESENVOLVIMENTO

FIAP

FÁBRICA DE DESENVOLVIMENTO – 2018

Prof. Thiago T. I. Yamamoto

#12 – DESIGN PATTERNS

 thiagoyama  thiagoyama@gmail.com

CONTEÚDO PROGRAMÁTICO

#12 – DESIGN PATTERNS

- Design Patterns
- Connection Factory
- Repository
- View Model

DESIGN PATTERNS

FIAP

- Muitos padrões de projeto não estão atrelados a alguma plataforma específica;
- Padrão de solução para problemas **repetitivos;**
- Constitui um poderoso instrumento que baseado na orientação a objetos podem maximizar a **qualidade e a produtividade de software;**
- Permite criar aplicações robustas com soluções já testadas e minimizar o impacto de alterações durante o desenvolvimento;

CONNECTION FACTORY

CONNECTION FACTORY

FIAP

- Para ter o controle sobre a construção de objetos, podemos utilizar o padrão de projetos **Factory**;
- Criar uma conexão é algo muito repetitivo e trabalhoso;
- Vamos criar uma **ConnectionFactory** que deve retornar uma instância de **IDbConnection**;

CONNECTION FACTORY - IMPLEMENTAÇÃO

FIAP

```
public class ConnectionFactory
{
    private string _connectionString =
        ConfigurationManager
        .ConnectionStrings["Exemplo"].ConnectionString;

    public IDbConnection CreateConnection()
    {
        return new SqlConnection(_connectionString);
    }
}
```

FIAP

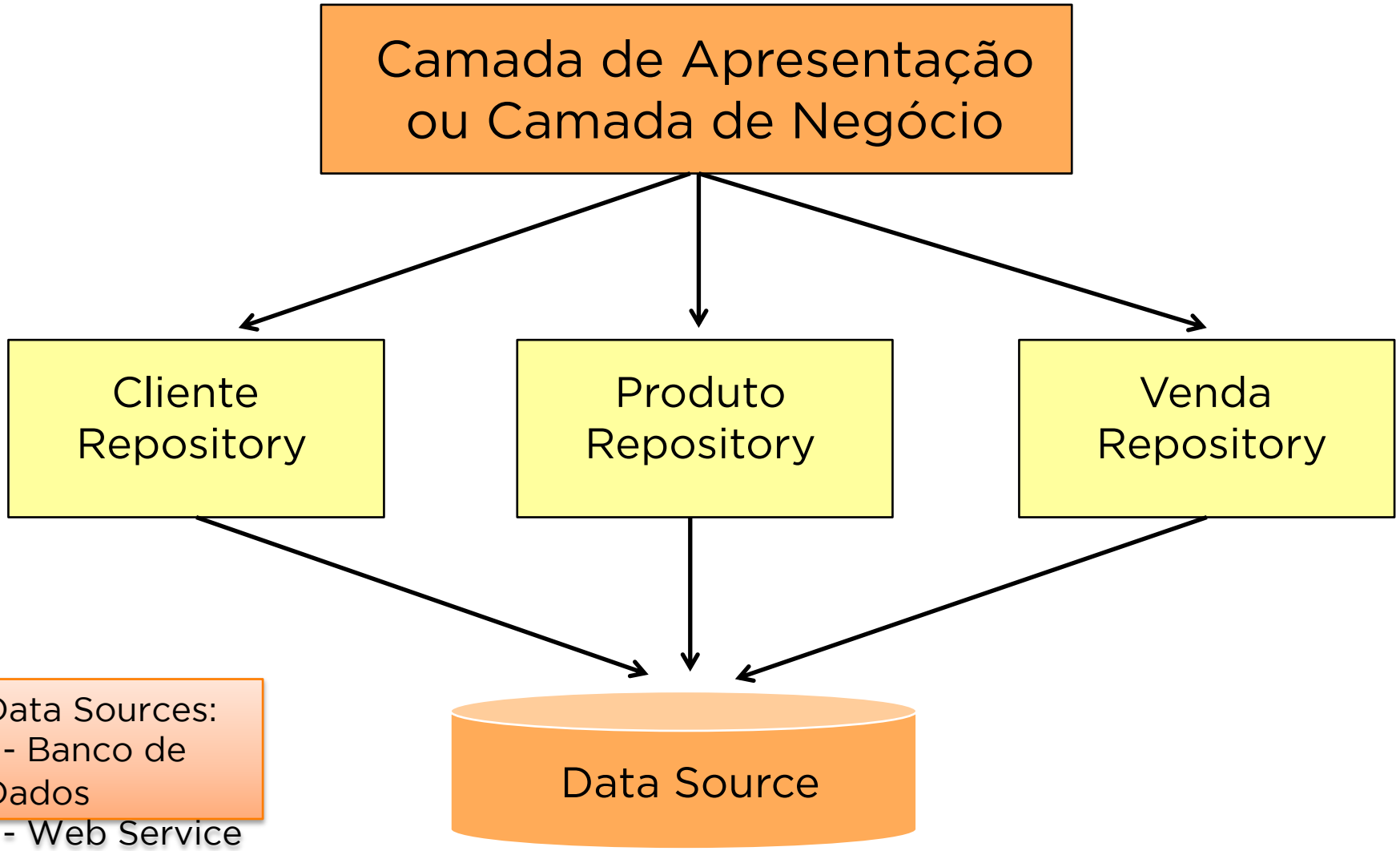
REPOSITORY

REPOSITORY

- Design Pattern para Acesso a Dados: permite realizar o isolamento entre a camada de acesso a dados (DAL) e a camada de apresentação ou camada de negócios.
- Benefícios:
  - Separação de conceitos;
  - Menos erros e códigos duplicados;
  - Facilita os testes automatizados;
  - Facilita a manutenção;

REPOSITORY

- Encapsula o acesso a dados:



REPOSITORY - INTERFACE

- Primeiramente, precisamos definir uma **interface** que será o contrato de acesso aos dados:

```
public interface IClienteRepository
{
    void Cadastrar(Cliente cliente);
    void Atualizar(Cliente cliente);
    void Remover(int id);
    Cliente Buscar(int id);
    IList<Cliente> Listar();
}
```

REPOSITORY - IMPLEMENTAÇÃO

- Depois, vamos construir a classe que irá implementar a interface **IClienteRepository**.
- Essa classe deve possuir uma ConnectionFactory, para criar as conexões.

```
public class ClienteRepository : IClienteRepository
{
    private ConnectionFactory _factory =
                                new ConnectionFactory();
}
```

REPOSITORY - ATUALIZAR

```
public void Atualizar(Cliente cliente)
{
    using (var db = _factory.CreateConnection())
    {
        string sql = "UPDATE Cliente SET Nome = @Nome,
        DataNascimento = @DataNascimento, Saldo =
        @Saldo, Especial = @Especial WHERE Id = @Id";
        db.Execute(sql, cliente);
    }
}
```

REPOSITORY - BUSCAR

FIAP

```
public Cliente Buscar(int id)
{
    using (var db = _factory.CreateConnection())
    {
        string sql = "SELECT * FROM Cliente WHERE id = @id";
        var cliente = db.Query<Cliente>(sql,
                        new { id = id }).SingleOrDefault();
        return cliente;
    }
}
```

REPOSITORY - CADASTRAR

```
public void Cadastrar(Cliente cliente)
{
    using (var db = _factory.CreateConnection())
    {
        string sql = "INSERT INTO Cliente (Nome,
        DataNascimento, Saldo, Especial) VALUES (@Nome,
        @DataNascimento, @Saldo, @Especial);" +
        "SELECT CAST(SCOPE_IDENTITY() as int)";

        int id = db.Query<int>(sql, cliente).Single();

        cliente.Id = id;

    }
}
```

REPOSITORY - LISTAR

```
public IList<Cliente> Listar()
{
    using (var db = _factory.CreateConnection())
    {
        var sql = "SELECT * FROM Cliente AS CLI INNER
        JOIN CarteiraMotorista AS CM ON CLI.Id =
        CM.ClienteId";

        IList<Cliente> lista = db.Query<Cliente,
        CarteiraMotorista, Cliente>(sql,
        (cliente, carteira) => {
        cliente.CarteiraMotorista = carteira; return
        cliente; }, splitOn: "Id,ClienteId").ToList();

        return lista;
    }
}
```

REPOSITORY - REMOVER

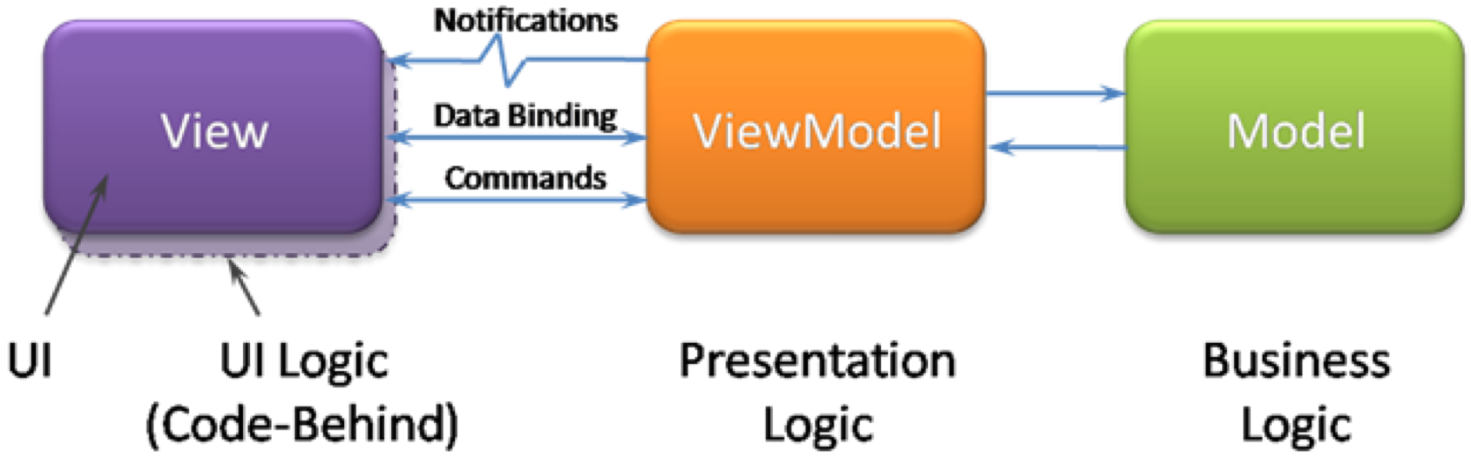
```
public void Remover(int id)
{
    using (var db = _factory.CreateConnection())
    {
        using (var txScope = new TransactionScope())
        {
            db.Execute("DELETE FROM Endereco WHERE
            ClienteId = @id", new { id = id });
            db.Execute("DELETE From Cliente WHERE id =
            @id", new { id = id });
            txScope.Complete();
        }
    }
}
```

FIAP

VIEW MODEL PATTERN

VIEW MODEL PATTERN

- Um padrão de Projetos que visa melhorar a organização do código e gestão das informações utilizados na View;
- Permite modelar várias entidades e informações que serão exibidas na View;



FIAP

O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis