

Professores Teoria: Bonato, Hamilton
Laboratório: Calvetti, Hamilton, Keity e Rodrigo

Aula: 3
Assunto: DAO, TO, Service e Factory

Conceitos Básicos

Padrões de Projeto (Design Patterns) são soluções usadas e experimentadas para problemas comuns do desenvolvimento de software. Os Design Patterns foram idealizados por Kent Beck, o mesmo da Extreme Programming, na década de 80. O primeiro pattern documentado do qual se tem notícia é o MVC, publicado em 88 por Karner & Pope no Journal of OOP como um padrão para a organização do código em SmallTalk. Mas o assunto ganhou fama mesmo quando Gamma, Helm, Johnson e Vlissides, a Gang of Four (GoF), publicou um livro com 23 design patterns, inspirando-se no trabalho do arquiteto (de prédios, não de software) Christopher Alexander.

Vocês já tiveram uma visão de patterns ao aprenderem em METDS, no ano passado, os patterns GRASP, de Craig Larman.

Os patterns devem ser descritos de uma maneira clara e organizada para facilitar seu uso. A GoF sugere o seguinte modelo, que não é obrigatório. Há outros modelos de descrição de patterns, mas todos contém mais ou menos as mesmas informações.

DAO - Data Access Object

Nome: Data Access Object

Intenção: toda a complexidade do acesso a dados deve ser abstraída das classes de negócio

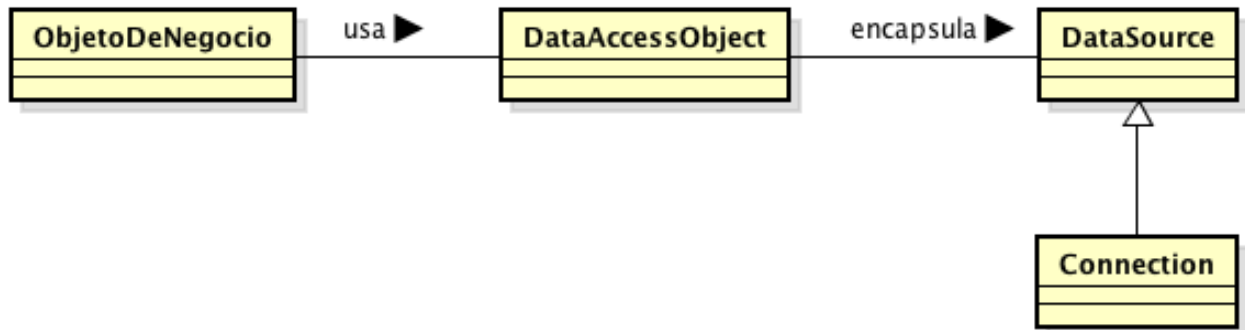
Apelido: DAO

Motivação:

- acessar dados em um banco depende muito do banco, do fabricante, do servidor, do driver de JDBC
 - os dados também podem estar em um flat file ou até um um arquivo XML
- A solução é criar um objeto para encapsular acesso ao banco; este objeto deve gerenciar o DataSource (Conexão com o Banco de Dados ou outras fontes de dados) e ter métodos CRUD

(create/retrieve/update/ delete) para inserir dados em tabelas, recuperá-los, atualizá-los ou deletá-los.

Estrutura:



Participantes e Responsabilidades:

ObjetoDeNegócio: quando quer persistir dados ou recuperar dados (operações CRUD) o ObjetoDeNegócio instancia ou obtém seu respectivo DAO e chama os métodos do DAO para fazer isso.

DataSource: representa a fonte de dados; se a fonte for um banco de dados, o objeto DataSource será uma conexão com o banco (Connection).

Colaborações:

1. O usuário solicita uma ação à GUI.
2. A classe de negócio correspondente é acionada.
3. A classe de negócio o instancia o DAO e pede a ele para persistir ou recuperar dados.
4. O DAO abre a conexão com o banco, realiza o que foi solicitado e fecha a conexão.

Consequências:

1. Toda a comunicação como o banco de dados fica transparente para a classe de negócio, que não precisa se preocupar com isso.
2. Porém note que nos métodos de persistência, principalmente o inserir e o atualizar, é necessário passar todos os dados da classe de negócio como para o DAO.
3. E, no método de recuperar os dados (select), é necessário usar um ArrayList não tipado (i.e., do tipo Object) para passar os dados solicitados de volta. Isso é perigoso, pois é necessário fazer castings no método de negócio. Veja no código que foi usada uma Anotação @SupressWarnings para remover as mensagens relativas ao perigo do uso de rawtypes e castings.
4. Obter a conexão também é uma tarefa complicada. Em vez do DAO instanciar a conexão, seria melhor outra classe fazer isso, pois ficaria mais flexível.

Implementação:

Usar em conjunto com o DAO os patterns TransferObject para resolver os problemas 2 e 3 acima e o pattern Factory para resolver o problema 4.

Código Exemplo: Classe DAO

```
package arqdesis_aula02;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class ClienteDAO {

    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    // Obtém conexão com o banco de dados
    public Connection obterConexao() throws SQLException {
        return DriverManager
            .getConnection("jdbc:mysql://localhost/tutorial?user=alunos&password=alunos");
    }

    public void incluir(int id, String nome, String fone) {
        String sqlInsert = "INSERT INTO cliente(id, nome, fone) VALUES (?, ?, ?)";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
            stm.setInt(1, id);
            stm.setString(2, nome);
            stm.setString(3, fone);
            stm.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void atualizar(int id, String nome, String fone) {
        String sqlUpdate = "UPDATE cliente SET nome=?, fone=? WHERE id=?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlUpdate);) {
            stm.setString(1, nome);
            stm.setString(2, fone);
            stm.setInt(3, id);
            stm.execute();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

public void excluir(int id) {
    String sqlDelete = "DELETE FROM cliente WHERE id = ?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obterConexao();
        PreparedStatement stm = conn.prepareStatement(sqlDelete);) {
        stm.setInt(1, id);
        stm.execute();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

@SuppressWarnings({ "rawtypes", "unchecked" })
public ArrayList carregar(int id) {
    ArrayList retorno = new ArrayList();
    String sqlSelect = "SELECT nome, fone FROM cliente WHERE cliente.id = ?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obterConexao();
        PreparedStatement stm = conn.prepareStatement(sqlSelect);) {
        stm.setInt(1, id);
        try (ResultSet rs = stm.executeQuery();) {
            if (rs.next()) {
                retorno.add(rs.getString("nome"));
                retorno.add(rs.getString("fone"));
            } else {
                retorno.add(null);
                retorno.add(null);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } catch (SQLException e1) {
        System.out.print(e1.getStackTrace());
    }
    return retorno;
}
}

```

Código Exemplo: Classe de Negócio

```

package arqdesis_aula02;

import java.util.ArrayList;

public class Cliente {
    private int id;
    private String nome;
    private String fone;

    public Cliente(int id, String nome, String fone) {
        this.id = id;
        this.nome = nome;
        this.fone = fone;
    }

    public int getId() {

```

```

        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getFone() {
        return fone;
    }

    public void setFone(String fone) {
        this.fone = fone;
    }

    public void criar() {
        ClienteDAO dao = new ClienteDAO();
        dao.incluir(id, nome, fone);
    }

    public void atualizar() {
        ClienteDAO dao = new ClienteDAO();
        dao.atualizar(id, nome, fone);
    }

    public void excluir() {
        ClienteDAO dao = new ClienteDAO();
        dao.excluir(id);
    }

    public void carregar() {
        ClienteDAO dao = new ClienteDAO();
        @SuppressWarnings("rawtypes")
        ArrayList dados = dao.carregar(id);
        nome = (String) dados.get(0);
        fone = (String) dados.get(1);
    }

    @Override
    public String toString() {
        return "Cliente [id=" + id + ", nome=" + nome + ", fone=" + fone + "];"
    }
}

```

Patterns relacionados: Facade, Transfer Object, Factory

TO - Transfer Object

Nome: Transfer Object

Intenção: transferir dados de negócio entre as várias camadas de uma aplicação

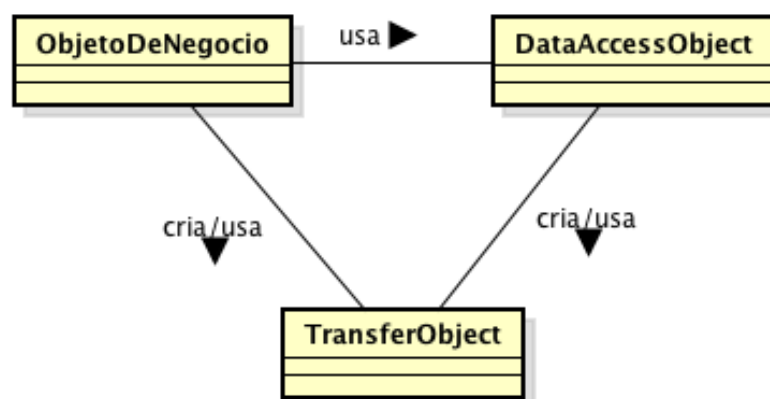
Apelido: TO

Motivação:

- todos os acessos entre camadas tem um grande overhead para transferência de dados pela rede
- geralmente um objeto de negócio precisa de uma grande quantidade de dados
- fazer diversas chamadas para recuperar dado por dado pode tornar a aplicação excessivamente lenta
- passar vários parâmetros em um método torna o código deslegante e difícil de manter
- retornar dados usando uma ArrayList de objects é uma má prática de programação

A solução é criar um objeto para encapsular os dados de negócio; preencher todos os dados em uma camada, transferir o objeto empacotando os dados de uma camada para outra e recuperar os dados; o TO deve implementar a interface `java.io.Serializable` para poder ser transferido entre camadas.

Estrutura:



Participantes e Responsabilidades

ObjetoDeNegócio: instancia um TO e passa para o DAO quando quer inserir, atualizar ou excluir dados; recebe um TO do DAO quando quer recuperar dados.

DataAccessObject: instancia um TO, preenche-o com os dados recuperados do banco e passa-o para o ObjetoDeNegócio; recebe o TO do ObjetoDeNegócio para inserir, atualizar ou excluir dados.

Colaborações:

Para fazer o CUD do CRUD

1. O usuário solicita uma inclusão, alteração ou exclusão à GUI.
2. A classe de negócio correspondente é acionada.
3. A classe de negócio instancia um TO e preenche-o com os dados corretos.
4. A classe de negócio o instancia o DAO e pede a ele para persistir os dados, passando o TO como parâmetro.
5. O DAO abre a conexão com o banco, realiza o que foi solicitado e fecha a conexão.

Para fazer o R do CRUD

1. O usuário faz uma consulta n GUI.
2. A classe de negócio correspondente é acionada.
3. A classe de negócio o instancia o DAO e pede a ele recuperar dados, passando o critério como parâmetro.
5. O DAO abre a conexão com o banco, faz o select, instancia um TO e preenche-o com os dados obtidos.
6. O DAO fecha a conexão e retorna o TO para a classe de negócio.
7. A classe de negócio recupera do TO dos dados passados e atualiza a GUI.

Consequências:

1. A passagem de parâmetros para o DAO fica simplificada.
2. Não é mais necessário passar um ArrayList genérico para a classe de negócio, mas agora é passada um objeto específico, o TO.
3. Duplicação de código entre a classe de negócio e o TO.

Implementação:

Usar em conjunto com o DAO e com a classe de negócio; porém é melhor usar a variante Service + Javabean (a seguir).

Código Exemplo: Classe TO

```
package arqdesis_aula02a;

public class ClienteTO {
    private int id;
    private String nome;
    private String fone;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```

    public String getFone() {
        return fone;
    }
    public void setFone(String fone) {
        this.fone = fone;
    }
}

```

Código Exemplo: Classe DAO

```

package arqdesis_aula02a;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class ClienteDAO {

    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    // Obtém conexão com o banco de dados
    public Connection obterConexao() throws SQLException {
        return DriverManager

.getConnection("jdbc:mysql://localhost/tutorial?user=alunos&password=alunos");
    }

    public void incluir(ClienteTO to) {
        String sqlInsert = "INSERT INTO cliente(id, nome, fone) VALUES (?, ?, ?)";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
            stm.setInt(1, to.getId());
            stm.setString(2, to.getNome());
            stm.setString(3, to.getFone());
            stm.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void atualizar(ClienteTO to) {
        String sqlUpdate = "UPDATE cliente SET nome=?, fone=? WHERE id=?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlUpdate);) {
            stm.setString(1, to.getNome());
            stm.setString(2, to.getFone());

```



```

        stm.setInt(3, to.getId());
        stm.execute();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void excluir(ClienteTO to) {
    String sqlDelete = "DELETE FROM cliente WHERE id = ?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obterConexao();
        PreparedStatement stm = conn.prepareStatement(sqlDelete);) {
        stm.setInt(1, to.getId());
        stm.execute();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public ClienteTO carregar(int id) {
    ClienteTO to = new ClienteTO();
    String sqlSelect = "SELECT nome, fone FROM cliente WHERE cliente.id = ?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obterConexao();
        PreparedStatement stm = conn.prepareStatement(sqlSelect);) {
        stm.setInt(1, id);
        try (ResultSet rs = stm.executeQuery();) {
            if (rs.next()) {
                to.setNome(rs.getString("nome"));
                to.setFone(rs.getString("fone"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } catch (SQLException e1) {
        System.out.print(e1.getStackTrace());
    }
    return to;
}
}

```

Código Exemplo: Classe de Negócio

```

package arqdesis_aula02a;

public class Cliente {
    private int id;
    private String nome;
    private String fone;

    public Cliente(int id, String nome, String fone) {
        this.id = id;
        this.nome = nome;
        this.fone = fone;
    }

    public int getId() {

```

```

        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getFone() {
        return fone;
    }

    public void setFone(String fone) {
        this.fone = fone;
    }

    public void criar() {
        ClienteDAO dao = new ClienteDAO();
        ClienteTO to = new ClienteTO();
        to.setId(id);
        to.setNome(nome);
        to.setFone(fone);
        dao.incluir(to);
    }

    public void atualizar() {
        ClienteDAO dao = new ClienteDAO();
        ClienteTO to = new ClienteTO();
        to.setId(id);
        to.setNome(nome);
        to.setFone(fone);
        dao.atualizar(to);
    }

    public void excluir() {
        ClienteDAO dao = new ClienteDAO();
        ClienteTO to = new ClienteTO();
        to.setId(id);
        dao.excluir(to);
    }

    public void carregar() {
        ClienteDAO dao = new ClienteDAO();
        ClienteTO to = dao.carregar(id);
        nome = to.getNome();
        fone = to.getFone();
    }

    @Override
    public String toString() {
        return "Cliente [id=" + id + ", nome=" + nome + ", fone=" + fone + "];"
    }

```

```
}  
}
```

Patterns relacionados: DataAccessObject, Factory, Service

Service

Nome: Service

Intenção: simplificar a classe de negócio e o TO evitando redundância de código

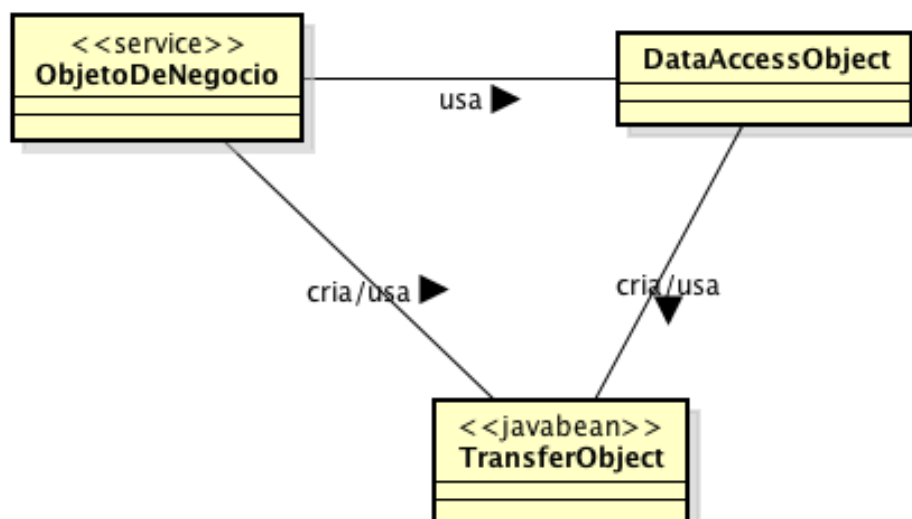
Apelido: Service

Motivação:

- os dados do Transfer Object e da Classe de Negócio são os mesmos, causando redundância e duplicação de código
- os frameworks Java como Spring e JSF esperam que o objeto de transferência de dados (TO) seja um javabeen com o mesmo nome da classe de negócio.

A solução é separar a classe de negócio em duas partes. Uma é o javabeen, que é igual ao TO mas sem o sufixo, tendo o nome da classe de negócio. A outra classe tem o nome da classe de negócio sufixada por Service. Ela contém os métodos de negócio que recebem o javabeen como parâmetro e/ou o retornam.

Estrutura:



Participantes e Responsabilidades

Service: instancia um Javabeen e passa para o DAO quando quer inserir, atualizar ou excluir dados; recebe um Javabeen do DAO quando quer recuperar dados.

DataAccessObject: instancia um Javabeen, preenche-o com os dados recuperados do banco e passa-o para o Service; recebe o Javabeen do Service para inserir, atualizar ou excluir dados.

Colaborações:

Semelhantes às do TO.

Consequências:

1. Elimina a redundância e a duplicação de código.
2. O Javabeen pode ser utilizado nos Frameworks.

Implementação:

Usar em conjunto com o DAO.

Código Exemplo: Classe Javabeen

```
package arqdesis_aula02a;

import java.io.Serializable;

public class Cliente implements Serializable {
    private int id;
    private String nome;
    private String fone;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getFone() {
        return fone;
    }
    public void setFone(String fone) {
        this.fone = fone;
    }
}
```

Código Exemplo: Classe DAO

```
package arqdesis_aula02a;
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

public class ClienteDAO {

    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    // Obtém conexão com o banco de dados
    public Connection obterConexao() throws SQLException {
        return DriverManager

.getConnection("jdbc:mysql://localhost/tutorial?user=alunos&password=alunos");
    }

    public void incluir(Cliente to) {
        String sqlInsert = "INSERT INTO cliente(id, nome, fone) VALUES (?, ?, ?)";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
            stm.setInt(1, to.getId());
            stm.setString(2, to.getNome());
            stm.setString(3, to.getFone());
            stm.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void atualizar(Cliente to) {
        String sqlUpdate = "UPDATE cliente SET nome=?, fone=? WHERE id=?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlUpdate);) {
            stm.setString(1, to.getNome());
            stm.setString(2, to.getFone());
            stm.setInt(3, to.getId());
            stm.execute();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void excluir(Cliente to) {
        String sqlDelete = "DELETE FROM cliente WHERE id = ?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = obterConexao();
            PreparedStatement stm = conn.prepareStatement(sqlDelete);) {

```

```

        stm.setInt(1, to.getId());
        stm.execute();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public Cliente carregar(int id) {
    Cliente to = new Cliente();
    String sqlSelect = "SELECT nome, fone FROM cliente WHERE cliente.id = ?";
    // usando o try with resources do Java 7, que fecha o que abriu
    try (Connection conn = obterConexao();
        PreparedStatement stm = conn.prepareStatement(sqlSelect);) {
        stm.setInt(1, id);
        try (ResultSet rs = stm.executeQuery();) {
            if (rs.next()) {
                to.setNome(rs.getString("nome"));
                to.setFone(rs.getString("fone"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } catch (SQLException e1) {
        System.out.print(e1.getStackTrace());
    }
    return to;
}
}

```

Código Exemplo: Service

```

package arqdesis_aula02a;

public class ClienteService {
    ClienteDAO dao;

    public Cliente() {
        dao = new ClienteDAO();
    }

    public void criar(Cliente to) {
        dao.incluir(to);
    }

    public void atualizar(Cliente to) {
        dao.atualizar(to);
    }

    public void excluir(Cliente to) {
        dao.excluir(to);
    }

    public Cliente carregar(int id) {
        Cliente to = dao.carregar(id);
        return to;
    }
}

```

```

@Override
public String toString() {
    return "Cliente [id=" + id + ", nome=" + nome + ", fone=" + fone + "]";
}
}

```

Patterns relacionados: DataAccessObject, Factory

Factory

Nome: Factory

Intenção: tornar a instanciação de objetos flexível.

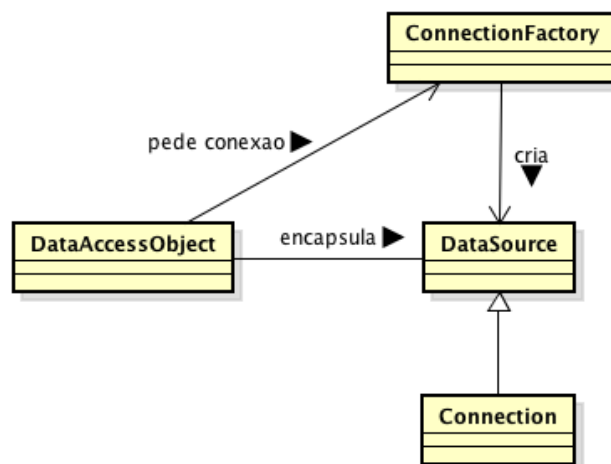
Apelido: não possui

Motivação:

- o objeto conexão muda de banco de dados para banco de dados
- o código SQL também varia um pouco
- o sistema tem que ser flexível para trabalhar com diferentes bancos de dados

A solução é criar uma classe delegar a instanciação dos objetos a ela.

Estrutura de uma fábrica de conexões:



Participantes e Responsabilidades

DataAccessObject: solicita a conexão à fábrica.

ConnectionFactory: tem um método static que instancia a conexão e a retorna a quem pediu.

Colaborações:

1. O objeto DAO solicita uma conexão ao objeto Fábrica.
2. O objeto Fábrica carrega o driver de JDBC correspondente, se ainda não o fez.
3. O objeto Fábrica cria a conexão solicitada e a retorna ao DAO.
4. O DAO realiza o que foi solicitado e fecha a conexão.

Consequências:

1. Toda a complexidade de abrir a conexão fica transparente para o DAO.

Implementação:

Usar métodos static para evitar a instanciação do Factory,

Código Exemplo para Criação de Factories de Conexões

```
package arqdesis_aula02b;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionFactory {
    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    // Obtém conexão com o banco de dados
    public static Connection obtemConexao() throws SQLException {
        return DriverManager

        .getConnection("jdbc:mysql://localhost/tutorial?user=alunos&password=alunos");
    }
}
```

Código Exemplo do DAO usando o ConnectionFactory

```
package arqdesis_aula02b;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```



```

import java.util.ArrayList;

public class ClienteDAO {

    public void incluir(ClienteTO to) {
        String sqlInsert = "INSERT INTO cliente(id, nome, fone) VALUES (?, ?, ?)";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = ConnectionFactory.obtemConexao();
             PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
            stm.setInt(1, to.getId());
            stm.setString(2, to.getNome());
            stm.setString(3, to.getFone());
            stm.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void atualizar(ClienteTO to) {
        String sqlUpdate = "UPDATE cliente SET nome=?, fone=? WHERE id=?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = ConnectionFactory.obtemConexao();
             PreparedStatement stm = conn.prepareStatement(sqlUpdate);) {
            stm.setString(1, to.getNome());
            stm.setString(2, to.getFone());
            stm.setInt(3, to.getId());
            stm.execute();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void excluir(ClienteTO to) {
        String sqlDelete = "DELETE FROM cliente WHERE id = ?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = ConnectionFactory.obtemConexao();
             PreparedStatement stm = conn.prepareStatement(sqlDelete);) {
            stm.setInt(1, to.getId());
            stm.execute();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public ClienteTO carregar(int id) {
        ClienteTO to = new ClienteTO();
        String sqlSelect = "SELECT nome, fone FROM cliente WHERE cliente.id = ?";
        // usando o try with resources do Java 7, que fecha o que abriu
        try (Connection conn = ConnectionFactory.obtemConexao();
             PreparedStatement stm = conn.prepareStatement(sqlSelect);) {
            stm.setInt(1, id);
            try (ResultSet rs = stm.executeQuery();) {
                if (rs.next()) {
                    to.setNome(rs.getString("nome"));
                    to.setFone(rs.getString("fone"));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

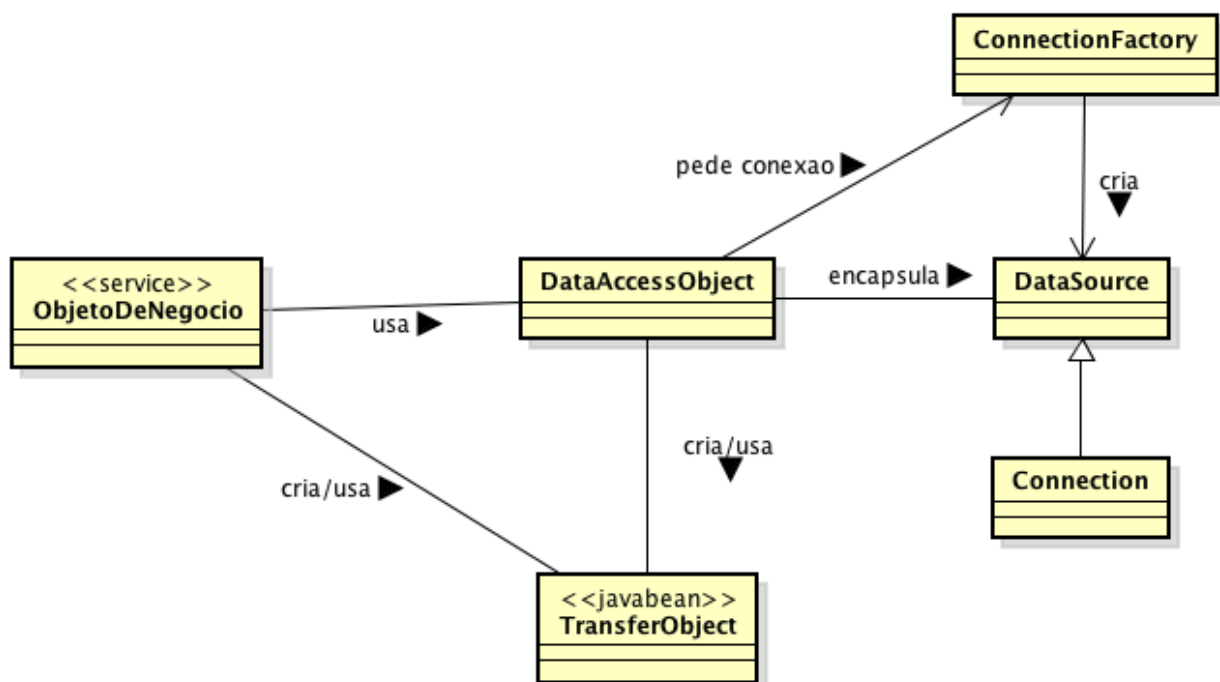
```

    }
    } catch (SQLException e1) {
        System.out.print(e1.getStackTrace());
    }
    return to;
}
}

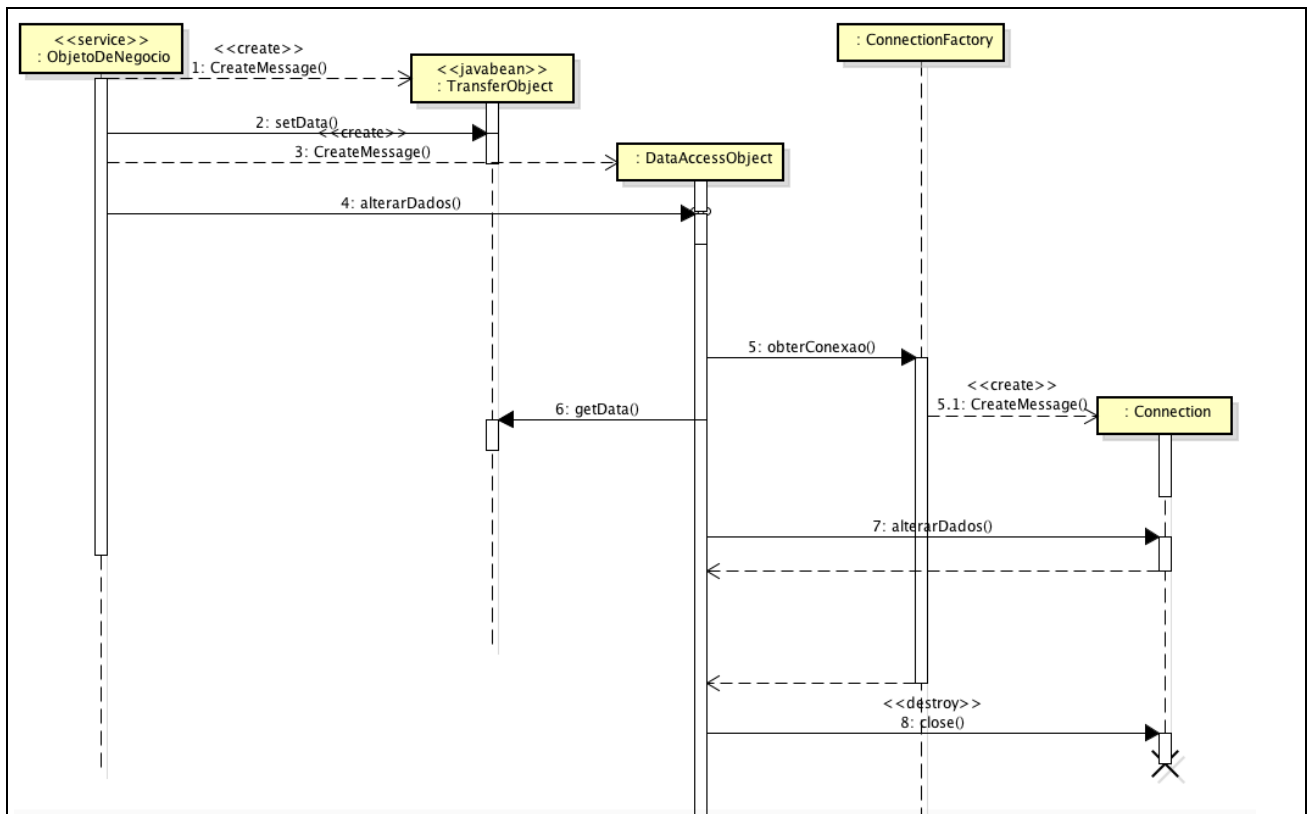
```

Como fica o uso dos 4 patterns juntos: Service, DAO, TO e Factory

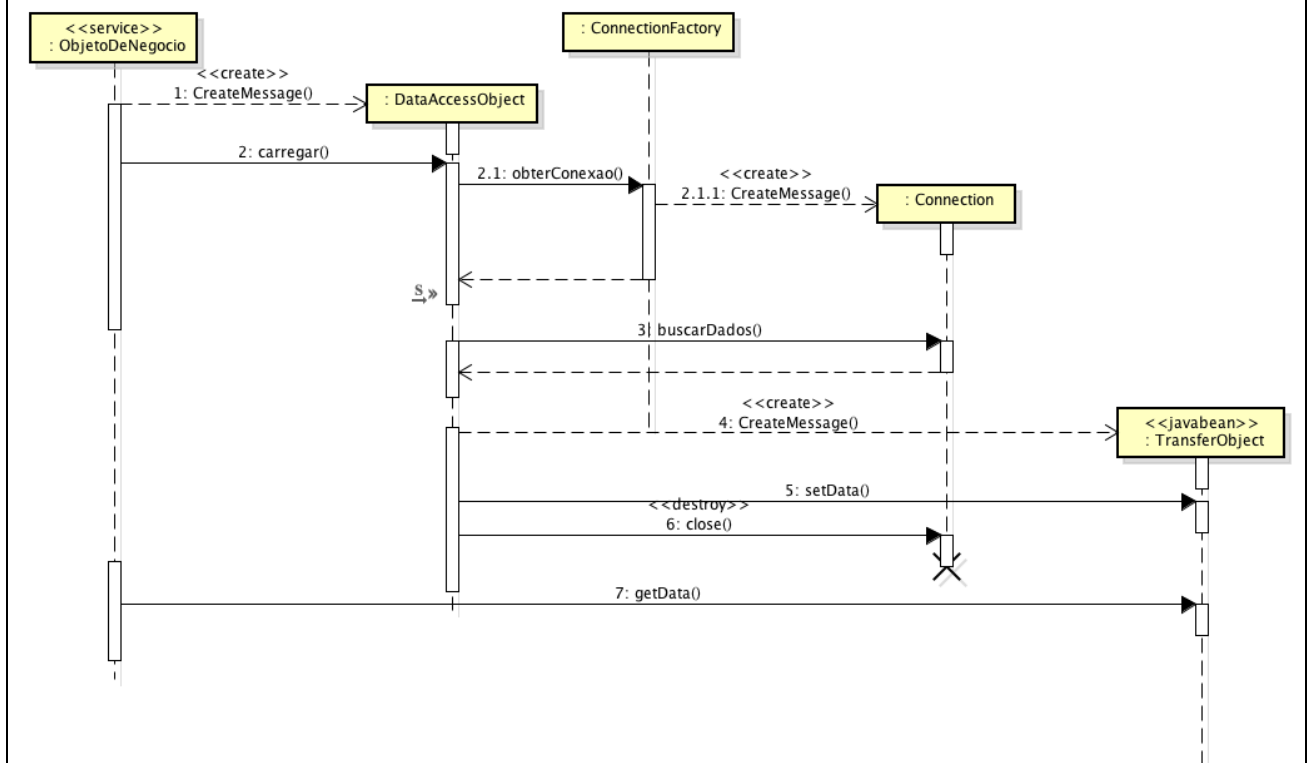
Estrutura:



Colaboração: fazer o CUD do CRUD



Colaboração: fazer o R do CRUD



Exercícios

1. Revisão da Teoria

- a. Crie um projeto Java no Eclipse. Ainda não precisa ser Web.
- b. Arraste as classes Cliente, ClienteDAO, ConnectionFactory, ClienteService e ClienteTest fornecidas e corrija eventuais erros com as packages. Não se esqueça de colocar o driver de JDBC no BuildPath.
- c. Rode os scripts de criação do banco e de carga de clientes no MySQL Workbench se necessário. Estes scripts foram fornecidos na aula 2.
- d. Execute o ClienteTest (clique sobre a classe no Project Browser com o botão direito, escolha Run As > JUnit Test) e veja se funcionou.
- e. Analise e entenda o código fonte.

2. Exercício para Nota (correção em aula)

A partir do código fonte do exercício da aula passada, onde foi criada a classe Pais, faça:

- a. Rode o teste do JUnit para garantir que está tudo ok
- b. Inicie a refatoração utilizando o pattern DAO. Rode novamente o JUnit para garantir que está tudo ok.
- c. Aplique o pattern Factory no DAO para a criação da conexão. Rode novamente o JUnit para garantir que está tudo ok.
- d. Aplique o pattern Service + Javabeen (TO). Agora você terá que adaptar a classe de teste do JUnit para que os testes funcionem novamente.

Bibliografia

ALUR, D.; CRUPI, J; MALKS, D.; **Core J2EE Patterns: Best Practices and Design Strategies**. Ed. Prentice Hall. 2a Edição. 2003. 528p.

Oracle; **Catálogo de Patterns JEE**; disponível online em <http://www.oracle.com/technetwork/java/catalog-137601.html>. Acessado em 21/02/2016

GAMMA, Erich. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos.** .
Porto Alegre: Bookman, 2000. 364 p. ISBN 8573076100

LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao processo unificado.** 3. ed. Porto Alegre: Bookman, 2007. 695 p. ISBN 9788560031528 (broch.)