Curso de Java

by Antonio Rodrigues Carvalho Neto

Data Access Object

Data Access Object - Definição

- É um Design Pattern da arquitetura corporativa, que busca minimizar o acoplamento da lógica de negócios com o código usado na persistência de dados.

Data Access Object - Contexto

- Acesso aos dados varia de acordo com a origem dos dados.
- Acesso ao mecanismos de armazenagem como bancos de dados varia amplamente conforme o tipo de armazenagem (relacional, orientado a objetos, arquivos diversos e outros), e também conforme o fornecedor da tecnologia.

Data Access Object - Problema

- O acesso aos dispositivos de persistência de dados como banco de dados, obriga o uso de linguagens e códigos adicionais para ler, transformar e gravar os dados.
- Tais códigos adicionais ainda podem variar conforme o tipo de persistência usado e conforme o fornecedor, submetendo o sistema a mudanças cada vez que existam alterações no tipo de persistência.

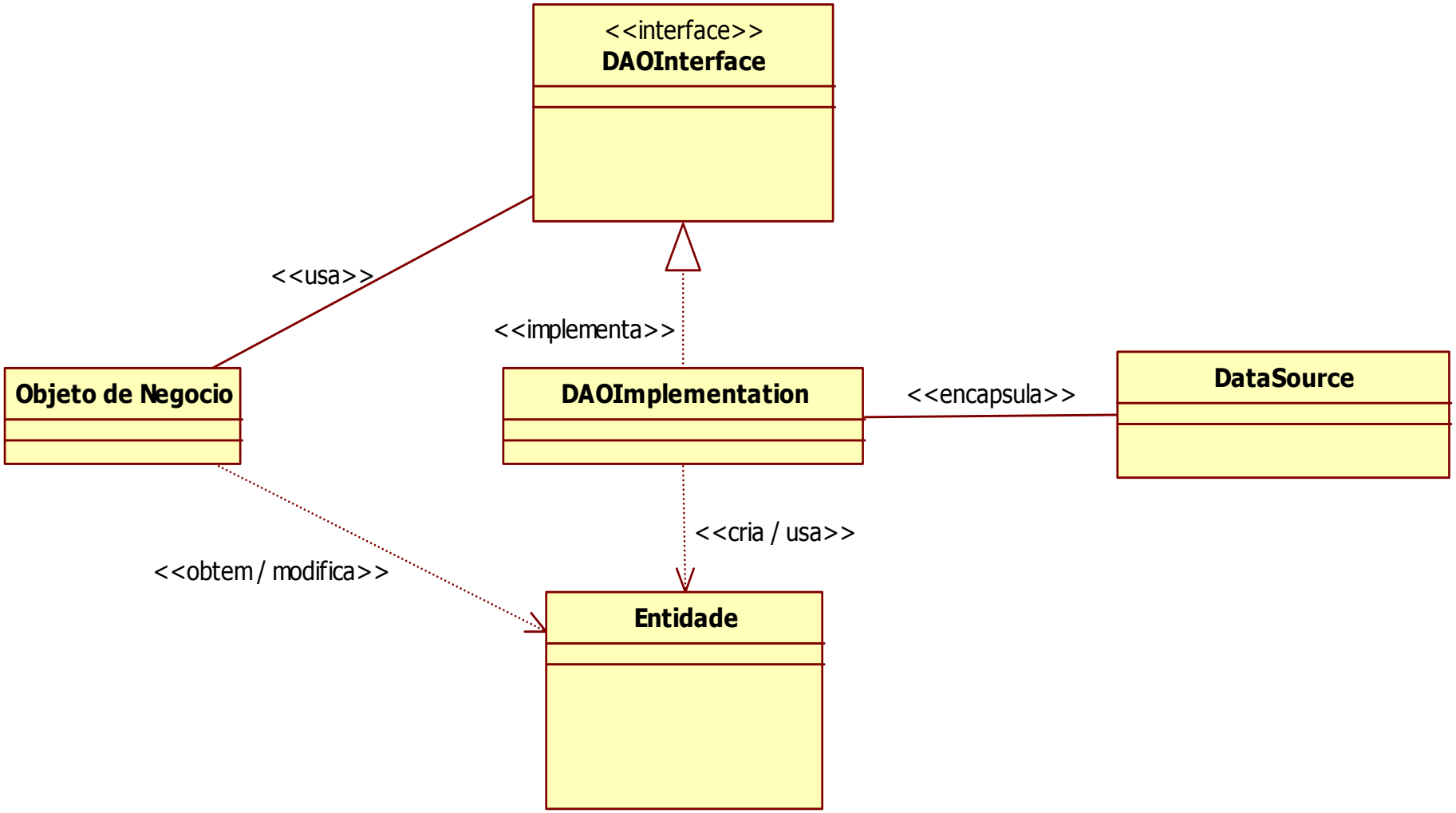
Data Access Object - Forças

- Varios componentes como Beans de Entidade, Beans de Sessão, Servlets e outros objetos precisam buscar e armazenar informações em sistemas de armazenamento como bancos de dados.
- APIs do sistema de armazenamento variam conforme o fornecedor e o tipo do sistema de armazenagem.
- Componentes normalmente usam APIs proprietárias para acessar fontes de dados external ou sistemas legados, (muitas vezes a API não usa os melhores recursos disponíveis)
- A portabilidade dos componentes é diretamente afetada quando os mecanismos de acesso e as APIs são contidos no componente.
- A fonte de dados ou o sistema de armazenamento precisam ser transparentes para os componentes de forma a assegurar uma possível mudança de produto, fornecedor, sistema de armazenamento e até mesmo do componente.

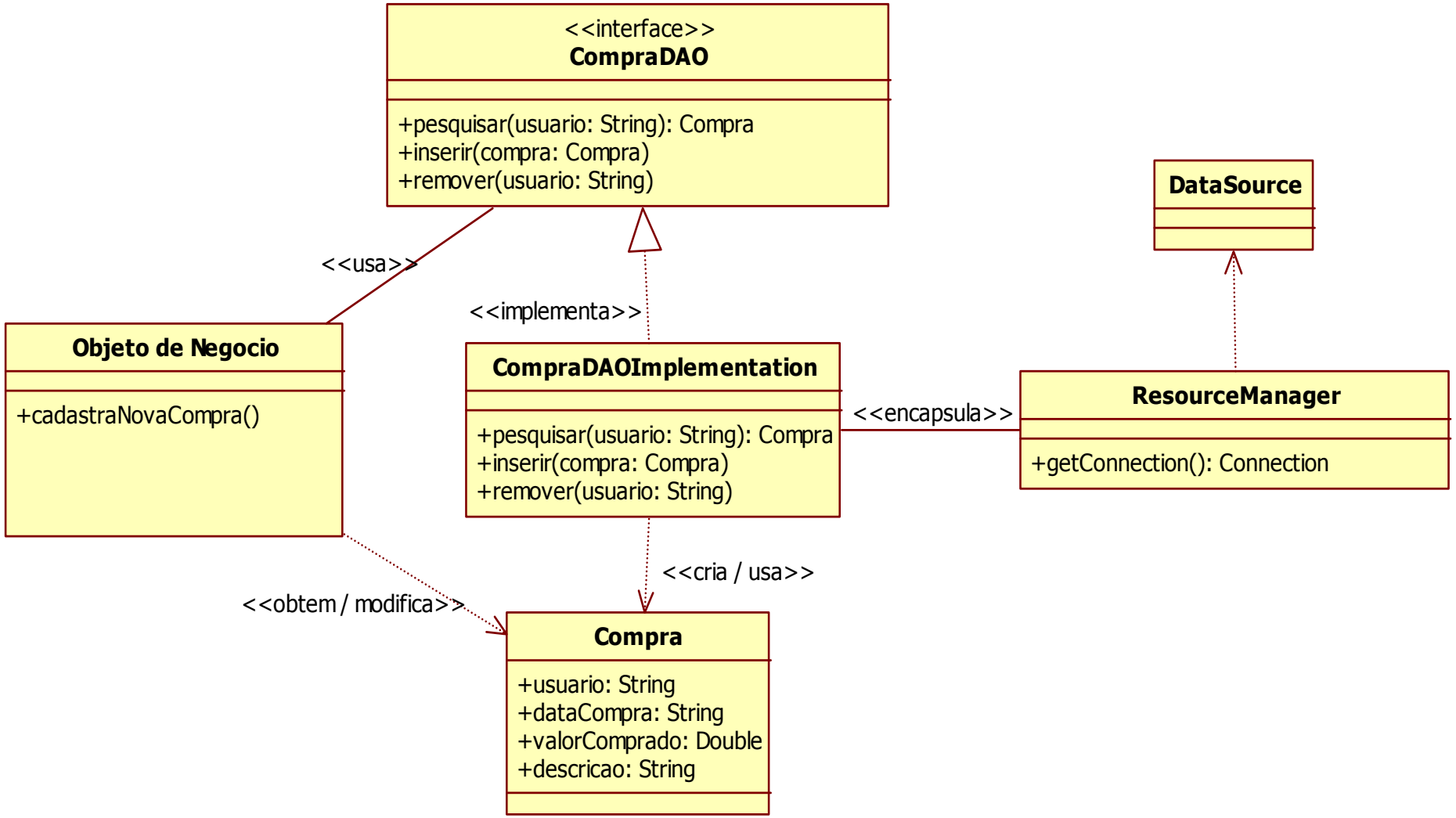
Data Access Object – Solução

- Usar o Data Access Object (DAO) para abstrair e encapsular todo o acesso as fontes de dados. O DAO gerenciará a conexão com a fonte de dados para ler e gravar informações.
- Dessa forma os componentes passam a acessar apenas interface do DAO
- A implementação desta interface esconde todos os detalhes da comunicação com a fonte de dados.
- Quando ocorrerem modificações na camada de dados apenas a implementação da interface DAO precisará ser alterada, evitando mudanças no componente que está conectado na interface e não na implementação.

Data Access Object – Estrutura



Data Access Object – Exemplo



Data Access Object - Códigos

- Código de exemplo da entidade :

```
public class Compra {
    String usuario;
    Calendar dataCompra;
    float valorComprado;
    String descricao;

    public String getUsuario() {
    return usuario;
    }
    public void setUsuario(String usuario) {
    this.usuario = usuario;
    }
    ....
}
```

Data Access Object - Códigos

- Código de exemplo para interface :

```
public interface CompraDAO {
        public Compra pesquisar(String usuario) throws SQLException;
        public void inserir(Compra compra) throws SQLException;
        public void remover (String usuario) throws SQLException;
}
```

Data Access Object - Códigos

- Código de exemplo para implementação:

```
public class CompraDAOImplementation implements CompraDAO {
    public Compra pesquisar(String usuario) throws SQLException {
        Connection con = ResourceManager.getConnection();
        String sql = "SELECT * FROM USUARIOS WHERE usuario = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setString(1, usuario);
        ResultSet rs = stmt.executeQuery();
        Compra c = null;
        if (rs.next()) {
            c = new Compra();
            c.setUsuario( rs.getString ("usuario") );
            Calendar cal = Calendar.getInstance();
            cal.setTime(rs.getDate("datacompra"));
            c.setDataCompra( cal );
            c.setValorComprado( rs.getFloat ("valorcomprado") );
            c.setDescricao( rs.getString ("descricao") );
        }
        ResourceManager.close(con);
        return c;
    }
    ......
}
```

Data Access Object - Códigos

- Código de exemplo para Resource:

```
public class ResourceManager
{  private static String JDBC_DRIVER   = "com.mysql.jdbc.Driver";
    private static String JDBC_URL      = "jdbc:mysql://localhost/bancoDados";
    private static String JDBC_USER     = "root";
    private static String JDBC_PASSWORD = "root";
    private static Driver driver = null;
    public static synchronized Connection getConnection() throws SQLException {
        if (driver == null) {
            try {
                Class jdbcDriverClass = Class.forName( JDBC_DRIVER );
                driver = (Driver) jdbcDriverClass.newInstance();
                DriverManager.registerDriver( driver );
            } catch (Exception e) {
                System.out.println( "Failed to initialise JDBC driver" );
                e.printStackTrace();
            }
        }
        return DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD );
    }
     public static void close(Connection conn) {
          try {
                if (conn != null) conn.close();
          } catch (SQLException sqle) {
                sqle.printStackTrace();
          }
     }
}
```

Bibliografia

- **ALUR, DEEPAK,** Core J2EE Patterns, Best pratices and design strategies - 1ª edição - pg. 371 à 388, Sun Microsystems
- **Core J2EE Pattern Catalog - http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html**