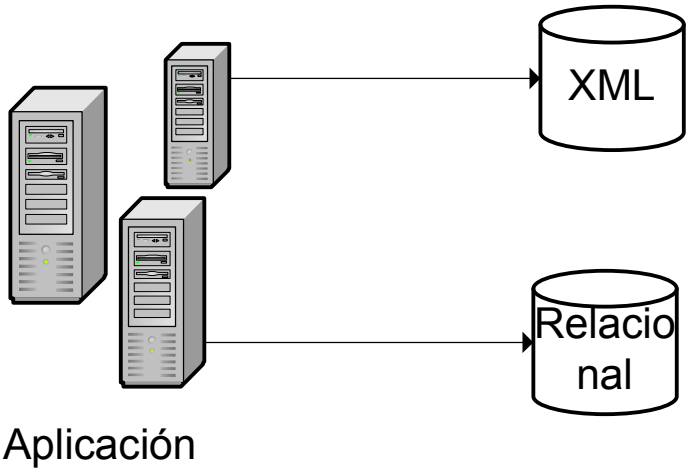
Nuevas Tecnologías de la Programación.

Patrón DAO.

Problema



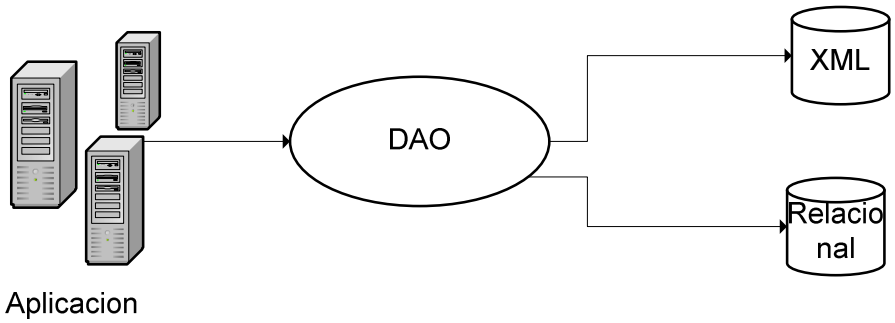
Problema

- A veces es necesario acceder a datos en diferentes repositorios de datos.
- Estos repositorios no tienen porque proporcionar una API común.
- En muchos de ellos es necesario tener en cuenta particularidades.
- Nuestra aplicación debe poder acceder de forma transparente a ellos.

Problema

¿Cómo hacerlo?

Solución

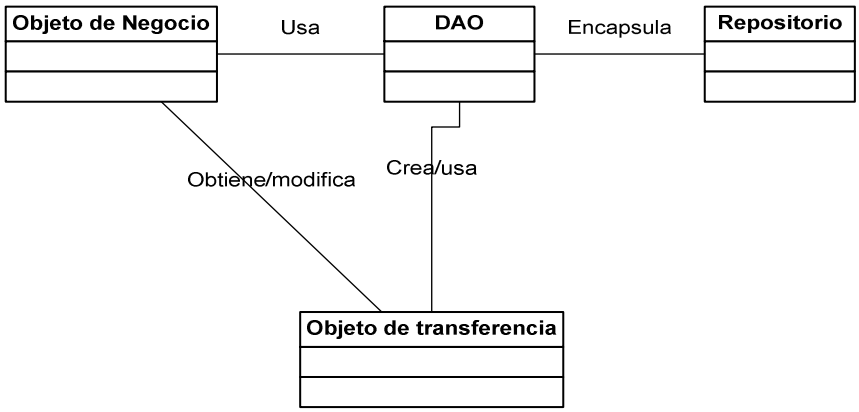


Solución

Usando el patrón DAO (Data Access Object).

DAO proporciona una interfaz única de acceso a los datos del repositorio independiente de cual sea este, permitiendo cambiarlo sin grandes cambios en el resto de la aplicación.

Modelo



Modelo

- **Objeto de negocio:** representa a la aplicación que desea acceder a los datos. Normalmente es algún tipo de objeto Java.
- **DAO:** principal elemento del patrón. Abstrae los detalles de implementación del repositorio al Objeto de negocio, permitiendo un acceso transparente a los datos. Además el Objeto de negocio delega en él las operaciones de carga y almacenamiento de datos.
- **Repositorio:** representa una implementación de un repositorio de datos. Suele ser una base de datos (Relacional, OO, XML, etc.). Aunque podría ser cualquier otro sistema (mainframe, servidor, etc.)
- **Objeto de transferencia:** se usa como transporte para los datos. DAO suele usarlo para devolver los datos al cliente. El cliente también podría usarlo para enviar al DAO los datos a actualizar en el repositorio.

Implementación.

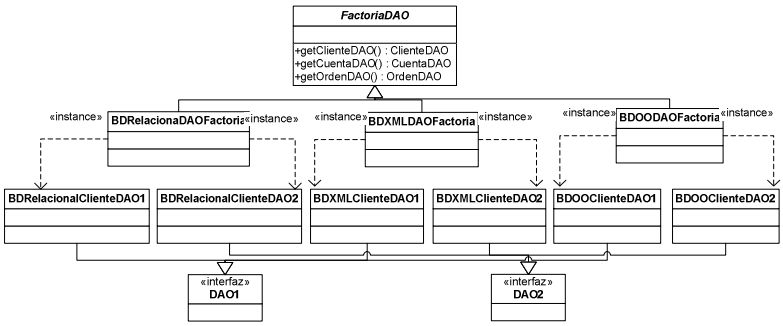
Para su implementación se usan principalmente dos patrones:

- Método factoría.
- Factoría abstracta.

Que permiten que este patrón goce de una gran flexibilidad.

El siguiente modelo muestra una posible implementación para tres repositorios diferentes (uno relacional, uno basado en XML y otro orientado a objetos) donde se usan dos accesos a datos diferentes.

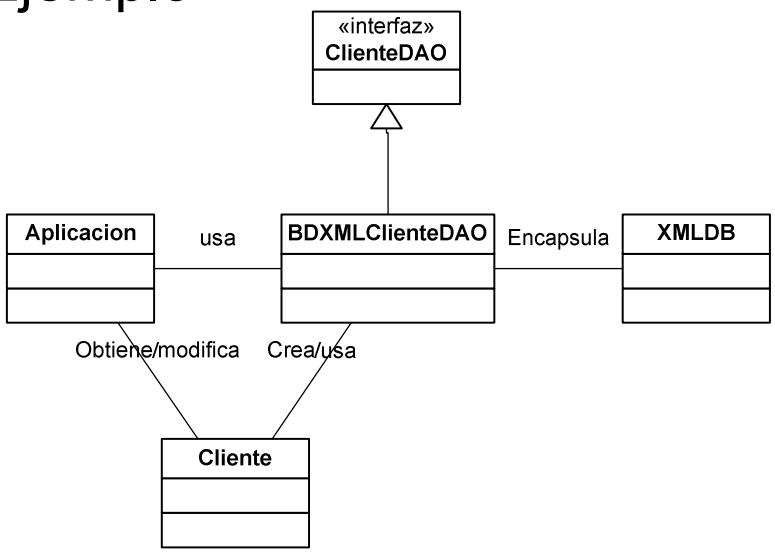
Implementación



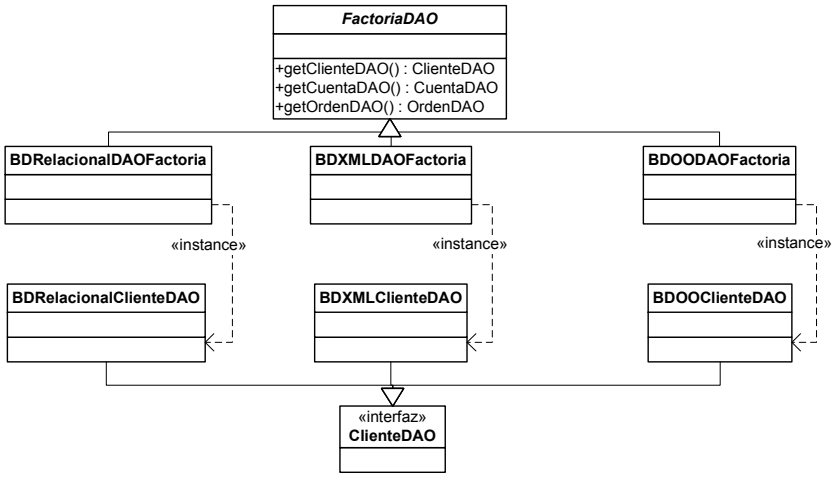
Ejemplo

El siguiente ejemplo muestra como se usa el patrón DAO para proporcionar acceso transparente a los datos sobre clientes.

Ejemplo



Ejemplo



Ejemplo

```
// Clase Abstracta FactoriaDAO
public abstract class FactoriaDAO {
   // Lista de los tipos DAO soportados por la factoria
   public static final int XML = 1;
   public static final int Relacional = 2;
   public static final int OO = 3;
   ...
   // Existe un metodo por cada DAO a crear.
   // Las factorias concretas son las responsables de
   implentarlos.
   public abstract ClienteDAO getClienteDAO();
   public abstract CuentaDAO getCuentaDAO();
   public abstract OrdenDAO getOrdenDAO();
    ...
```

Ejemplo

```
   public static DAOFactoria getDAOFactoria( int queFactoria) {
        switch (queFactory) {
                case XML:
                       return new BDXMLDAOFactoria();
                case Relacional:
                       return new BDRelacionalDAOFactory();
                case SYBASE:
                       return new BDOODAOFactory();
                ...
                default :
                       return null;
         }
   }
}
```

Ejemplo

```
// Implementacion de DAOFactoria usando el motor
// relacional Cloudscape
import java.sql.*;

public class BDRelacionalDAOFactoria extends DAOFactoria {

public static final String DRIVER=
   "COM.cloudscape.core.RmiJdbcDriver";
public static final String DBURL=
   "jdbc:cloudscape:rmi://localhost:1099/CoreJ2EEDB";

// método para crear una conexión Cloudscape

public static Connection crearConexion() {
 // Usar DRIVER y DBURL para crear una conexion
}
```

Ejemplo

```
public ClienteDAO getClienteDAO() {
   //BDRelacionalClienteDAO implementa ClienteDAO
   return new BDRelacionalClienteDAO();
}

public CuentaDAO getCuentaDAO() {
   //BDRelacionalCuentaDAO implementa CuentaDAO
   return new BDRelacionalCuentaDAO();
}

public OrdenDAO getOrdenDAO() {
   //BDRelacionalOrdenDAO implementa OrdenDAO
   return new BDRelacionalOrdenDAO();
}
 ...
}
```

Ejemplo

```
// Interfaz que deben implementar todos los ClienteDAOs

public interface ClienteDAO {
public int anadirCliente(...);
public boolean borrarCliente(...);
public Cliente encontrarCliente(...);
public boolean actualizarCliente(...);
public RowSet seleccionarClientesRS(...);
public Collection selecinarClientesTO(...);
 ...
 }
```

Ejemplo

```
// BDRelacionalClienteDAO implenta la interfaz ClienteDAO
// Esta clase contiene todo el codigo especifico de Cloudscape y SQL
// ocultando al cliente los detalles de implementación

import java.sql.*;
public class BDRelacionalClienteDAO implements ClienteDAO {

public CloudscapeCustomerDAO() {
    // inicializacion
}
// Los siguiente metodos pueden usar
// BDRelacionalClienteDAOFactoria.crearConexion()
// para establecer una conexión cuando lo necesiten

public int anadirCliente(...) {
    // Implementa añadir un cliente a la BD.
    // Devuelve el codigo del nuevo cliente
    // o -1 si hay error.
}
```

Ejemplo

```
public boolean borrarCliente(...) {
    // Implemente borrar un cliente
    // Devuelve true si es exitoso o false si no lo es
}

public Customer encontrarCustomer(...) {
// Implementa encontrar un cliente
// usando los parametros como criterio de búsqueda
// Devuelve el Objeto de transferencia si lo encuentra
// o null si no lo encuentra o hay un error
}

public boolean actualiazrCliente(...) {
// Implementa actualizar un cliente usando
// los datos del Objeto de transferencia que se
// Devuelve true si tiene éxito o false
// si no lo tiene o hay un error
}
```

Ejemplo

```
public RowSet seleccionarClientesRS(...) {
   // Implementa el buscar clientes
   // usando el criterio pasado como
   // argumento.
   // Devuelve el RowSet resultante.
}

public Collection seleccionarClientesTO(...)
{
   // Implementa el buscar clientes
   // usando el criterio pasado como
   // argumento.
   // Devuelve el resultando como una colección de Objetos
   // de transferencia
}
...
}
```

Ejemplo

```
public class Cliente{
// variables de instancia
int numeroCliente;
String nombre;
String direccion;
String ciudad;
...

// métodos get y set
...
...
}
```

Ejemplo

```
...
// Crear el DAOFactoria correspondiente
DAOFactoria relacionalFactoria =
DAOFactoria.getDAOFactoria(DAOFactoria.R);

// Crear un DAO
ClienteDAO cliDAO =
relacionalFactoria.getClienteDAO();

// Añadir un nuevo cliente
int newCustNo = cliDAO.anadirCliente(...)

//Encontrar un objeto cliente. Caputar el Objeto de
//transferencia.
Cliente cli = cliDAO.encontrarCliente(...);

//Modificar los valores del Objeto de transferencia
cli.setDireccion (...);
cli.setCorreo (...);
```

Ejemplo

```
...
// Crear el DAOFactoria correspondiente
DAOFactoria relacionalFactoria =
DAOFactoria.getDAOFactoria(DAOFactoria.R);

// Crear un DAO
ClienteDAO cliDAO =
relacionalFactoria.getClienteDAO();

// Añadir un nuevo cliente
int numeroClienteNuevo = cliDAO.anadirCliente(...)

//Encontrar un objeto cliente. Caputar el Objeto de
//transferencia.
Cliente cli = cliDAO.encontrarCliente(...);

//Modificar los valores del Objeto de transferencia
cli.setDireccion (...);
cli.setCorreo (...);
```

Ejemplo

```
// Actualizar el objeto cliente usando el DAO
cliDAO.actualizarCliente(cli);

// Borrar un objeto cliente
cliDAO.borrarCliente(…);

//Seleccionar todos los clientes en la misma ciudad
Clinete criterio = new Cliente();
Criterio.setCiudad(“Huelva”);
Collection listaClientes = cliDAO.seleccionarClientesTO(criterio);

//Devuelve una colección de Objetos de transferencia.
//Iterar a través de la colección para obtener los valores.
```