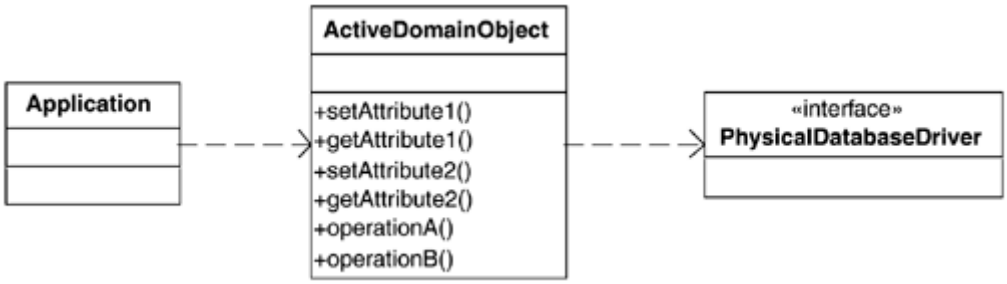
Asignatura programación de aplicaciones

Patrón Active Domain Object

Estructura



Ejemplo de código

```
public class Customer {

    // SQL statements for data access.
    private static final String QUERY
        = "SELECT * FROM CUSTOMERS WHERE CUSTID = ?";

    private static final String INSERT
        = "INSERT INTO CUSTOMERS "
        + "(LAST, FIRST, ADDRESS, CITY, STATE, COUNTRY, "
        + "ZIP, CUSTID) VALUES(?, ?, ?, ?, ?, ?, ?, ?)";

    private static final String UPDATE
        = "UPDATE CUSTOMERS SET LAST = ?, FIRST = ?, "
        + "ADDRESS = ?, CITY = ?, STATE = ?, COUNTRY = ?, "
        + "ZIP = ? WHERE CUSTID = ?";

    // The customer's data attributes.
    private int id;
    private String name;
    private Address address;

    // Indicates whether the data in the active domain
    // object matches that in the database.
    private boolean saved = false;

    /**
    Constructs a Customer object.  This constructor is
    intended for application code that adds new customers
    to the database.  It leaves the saved flag false to
    indicate that the data is not stored in the database
    yet.
    */
    public Customer(int id, String name, Address address) {
```

```
        this.id = id;
        this.name = name;
        this.address = address;
    }

    /**
    Constructs a new Customer object.  This constructor
    is intended for application code to read information
    for a single customer.
    */
    public Customer(int id) throws DataException {
        this.id = id;

        try {
            Connection connection
                = ConnectionFactory.getConnection();
            PreparedStatement statement
                = connection.prepareStatement(QUERY);

            statement.setInt(1, id);
            ResultSet resultSet = statement.executeQuery();
            resultSet.next();

            String lastName = resultSet.getString("LAST");
            String firstName = resultSet.getString("FIRST");
            String address = resultSet.getString("ADDRESS");
            String city = resultSet.getString("CITY");
            String state = resultSet.getString("STATE");
            String country = resultSet.getString("COUNTRY");
            String zip = resultSet.getString("ZIP");

            resultSet.close();
            statement.close();
            connection.close();

            initialize(id, lastName, firstName, address,
                       city, state, country, zip);
        }
        catch(SQLException e) {
            throw new DataException("Unable to read customer "
                + id, e);
        }
    }

    /**
    Constructs a Customer object.  This constructor is
    called only by CustomerList as it populates its
    contents.
    */
    public Customer(int id,
                    String lastName,
                    String firstName,
                    String address,
                    String city,
                    String state,
                    String country,
                    String zip) {

        initialize(id, lastName, firstName, address,
            city, state, country, zip);
    }
```

```
    /**
    Initializes the contents of this object based on
    physical data.  This is the mapping of data from
    its relational form to its domain object form.
    */
    private void initialize(int id,
                            String lastName,
                            String firstName,
                            String address,
                            String city,
                            String state,
                            String country,
                            String zip) {

        // Combine the first and last names, since
        // that is how the application
        // needs them.
        StringBuffer buffer = new StringBuffer();
        buffer.append(lastName);
        buffer.append(", ");
        buffer.append(firstName);
        name = buffer.toString();

        // Combine the city and state, since
        // that is how the application
        // needs them.
        buffer = new StringBuffer();
        buffer.append(city);
        buffer.append(", ");
        buffer.append(state);
        String cityState = buffer.toString();

        // Initialize the address.
        this.address = new Address();
        this.address.setAddress1(address);
        this.address.setCityState(cityState);
        this.address.setCountry(country);
        this.address.setPostalCode(zip);

        // Set this to true, since this information
        // was read from the database.
        saved = true;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }
```

```
    /**
    Saves the customer information to the database.
    This is the mapping of data from its domain object
    form to its relational form.  It uses the saved flag
    to determine whether to insert new data or update
    existing data.
    */
    public void save() throws DataException
    {
        // Split the name into first and last, since
        // the CUSTOMERS table stores these items in separate
        // columns.
        String first;
        String last;
        int comma = name.indexOf(',');
        if (comma >= 0) {
            first = name.substring(comma + 1).trim();
            last = name.substring(0, comma).trim();
        }
        else {
            first = "";
            last = name;
        }

        // Split the city and state, since the CUSTOMERS table
        // stores these items in separate columns.
        String cityState = address.getCityState();
        String city = "";
        String state = "";
        if (cityState != null) {
            comma = cityState.indexOf(',');
            if (comma >= 0) {
                city = cityState.substring(0, comma).trim();
                state = cityState.substring(comma).trim();
            }
            else {
                city = cityState;
            }
        }

        try {
            Connection connection
                = ConnectionFactory.getConnection();

            // If the customer information came from
            // the database, then issue an update.
            if (saved) {
                PreparedStatement statement
                    = connection.prepareStatement(UPDATE);
                statement.setString(1, last);
                statement.setString(2, first);
                statement.setString(3, address.getAddress1());
                statement.setString(4, city);
                statement.setString(5, state);
                statement.setString(6, address.getCountry());
                statement.setString(7,
                    address.getPostalCode());
                statement.setInt(8, id);
                statement.executeUpdate();
                statement.close();
            }
```

```
            // If the customer information did not
            // come from the database, then issue an insert.
            else {
                PreparedStatement statement
                    = connection.prepareStatement(INSERT);
                statement.setString(1, last);
                statement.setString(2, first);
                statement.setString(3, address.getAddress1());
                statement.setString(4, city);
                statement.setString(5, state);
                statement.setString(6, address.getCountry());
                statement.setString(7,
                    address.getPostalCode());
                statement.setInt(8, id);
                statement.executeUpdate();
                statement.close();

                // Indicate that the information now exists
                // in the database.
                saved = true;
            }

            connection.close();
        }
        catch(SQLException e) {
            throw new DataException("Unable to save customer "
                + id, e);
        }
    }
}
```

Clase CustomerList

```
public class CustomerList
{
    private static final String QUERY
        = "SELECT * FROM CUSTOMERS";

    private List contents = new LinkedList();

    /**
    Constructs a CustomerList object that represents
    every customer in the database.
    */
    public CustomerList() throws DataException {

        try {
            Connection connection
                = ConnectionFactory.getConnection();
            PreparedStatement statement
                = connection.prepareStatement(QUERY);
            ResultSet resultSet = statement.executeQuery();
            while(resultSet.next()) {
```

```
                int id = resultSet.getInt("CUSTID");
                String lastName = resultSet.getString("LAST");
                String firstName
                    = resultSet.getString("FIRST");
                String address
                    = resultSet.getString("ADDRESS");
                String city = resultSet.getString("CITY");
                String state = resultSet.getString("STATE");
                String country
                    = resultSet.getString("COUNTRY");
                String zip = resultSet.getString("ZIP");

                Customer customer = new Customer(id, lastName,
                    firstName, address, city, state, country,
                    zip);
                contents.add(customer);
            }
            resultSet.close();
            statement.close();
            connection.close();
        }
        catch(SQLException e) {
            throw new DataException(
                "Unable to read customer list", e);
        }
    }

    /**
    Returns an iterator that lists every customer in the list.
    */
    public Iterator iterator() {
        return contents.iterator();
    }
}
```