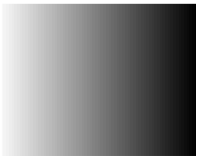


Data Access Object (DAO) Tasarım Şablonları Serisi

KurumsalJava.com

Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

Bu makale Özcan Acar tarafından yazılmış olan Java Tasarım Şablonları ve Yazılım Mimarileri isimli kitaptan alıntıdır.

Detaylı bilgiyi http://www.pusula.com adresinden edinebilirsiniz.

Data Access Object (DAO) Tasarım Şablonu

Birçok programın var olma nedeni veriler üzerinde işlem yapmak,verileri bilgibankalarında depolamak ve bu verileri tekrar edinmektir. Bu böyle olunca, verilerin program tarafından nasıl bilgibankalarına konulduğu ve tekrar edinildiği önem kazanmaktadır. Data Access Objects (DAO) tasarım şablonu ile, kullanılan veritabanına erişim ve veri depolama-edinme işlemi daha soyutlaştırılarak, diğer katmanların veritabanına olan bağımlılıkları azaltılır. DAO ile diğer katmanlar etkilenmeden veritabanı ve bilgibankası değiştirilebilir. Daha öncede belirttiğim gibi, amacımız birbirini kullanan ama birbirine bağımlılıkları çok az olan katmanlar oluşturmak ve gerekli olduğu zaman bir katmanı, diğer katmanlar etkilenmeden değiştirebilmek olmalıdır. Katmanlar arası bağımlılık interface sınıfları üzerinden olduğu sürece bu amacımıza her zaman ulaşabiliriz.

Belki inanmayabilirsiniz, lakin aşağıda yer alan JSP sayfasına birçok kurumsal denebilecek önemli projelerde bile rastlamak mümkündür. Zaman yetersizliğinden dolayı birçok programcı, kendilerine verilen görevleri aşağıdaki şekilde çözmek zorunda kalabilirler. Bu programcının hatası değildir! Proje menajerinin, yapılacak görevler için yeterince zaman ayrılmasına dikkat etmesi gerekmektedir. Doğru tahminler üzerinde kurulu olmayan bir proje planının çalışmamazlılığının acısı programcı ekipten çıkartılmamalıdır!

```
...
<%
Connection con = getConnection();
PreparedStatement pstmt = con.prepareStatement("select.....");
ResultSet rs = pstmt.executeQuery();
While(rs.next())
{
    Out.print(rs.getString(1));
}
rs.close();
pstmt.close();
con.close();
%>
```

Bu örnekte yapılmaması gereken hatalar bulunmaktadır. Öncelikle bunları inceleyelim:

- Bir JSP sayfası kesinlikle bilgibankasına bağlanıp, verileri edinmemelidir. JSP sayfası başka bir katmandan edinilen verileri sadece göstermekle yükümlüdür. Bir JSP sayfası içine yukarda yer aldığı şekilde gösterim mantığı harici başka kod eklendiğinde, JSP sayfasının bakımı ve geliştirilmesi güçleşir.
- Try-catch-finally bloğu kullanılmamıştır. Bilgibankasına bağlanmak için bir Connection nesnesi oluşturulmakta ve con.close() ile kapatılmaktadır. getConnection() operasyonundan sonra bir hata (Exception) oluşması durumunda con.close() satırına hiçbir zaman ulaşılamayacak ve bilgibankasına olan bağlantı kapatılamayacaktır! Bilgibankasına olan bağlantı con.close() ile kapatılmadığı sürece, diğer kullanıcılar tarafından kullanılamaz hale gelir.
- Bilgibankasında yer alan veriler bir ORM[1] (object relational mapping) framework ile nesnelere dönüştürülmeli ve bu nesneler kullanılmalıdır.

[1] Bakınız: http://en.wikipedia.org/wiki/Object-relational_mapping

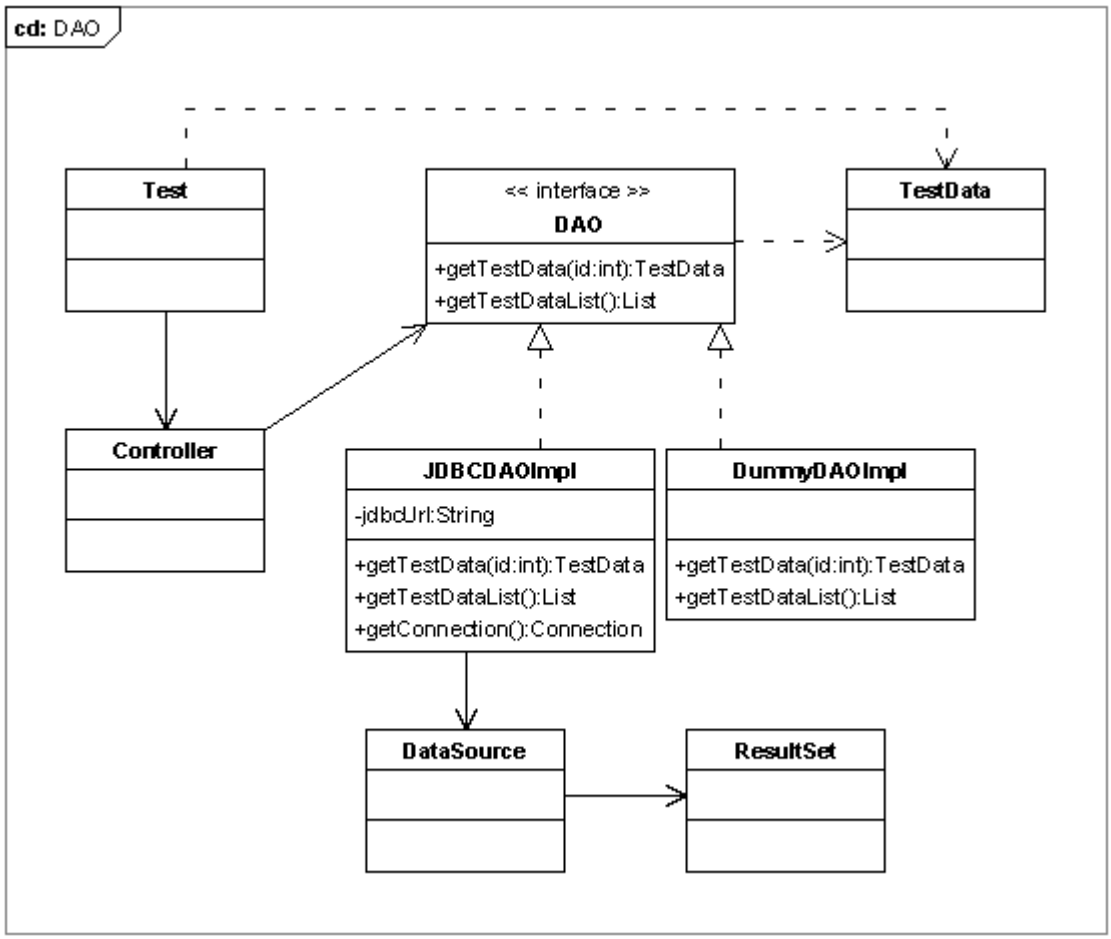
JSP sayfasının görevi sadece verileri kullanıcıya göstermek ise, bilgibankasına bağlanma ve diğer işlemleri bir Controller sınıfı halletmelidir. Örnek aşağıdaki şekilde değiştirilebilir:

```
...
<%
Controller controller = new ApplicationController();
List list = controller.getMemberList();

for(int i=0; i < list.size(); i++)
{
    Out.print(((Member)list.get()).getName());
    Out.print("<p>");
}
%>
...
```

ApplicationController sınıfı arka planda JDBC yada başka bir teknoloji kullanarak gerekli verileri bilgibankasından edinir. JSP sayfası getMemberList() metodu üzerinden bu verilere ulaşır. Bu şekilde çok basitte olsa iki katmanlı bir mimari oluşturarak, JSP sayfasının sadece gösterimden, Controller sınıfının bilgibankası işlemlerinden sorumlu olmasını sağladık.

Bu açıklamaların ardından DAO tasarım Şablonu kullanımını yakından inceleyelim.



DAO tasarım şablonunda uygulanmak istenen veri işlemleri (metotlar) DAO ismini (başka bir isimde olabilir) taşıyan bir interface sınıf bünyesinde toplanır. Bilgibankası ya da kullanılan başka bir veri kaynağı üzerinde işlem yapmak isteyen sınıf veya katmanlar (UML diagramında Controller sınıfı) sadece DAO interface sınıfını muhatap olarak kabul ederek, işlemleri yaparlar. Hangi DAO implementasyonun kullanıldığını DAO interface sınıfını kullananlar bilmek zorunda değildir.

Kullanılan veri erişim teknolojine göre, bu JDBC ya da JDBC teknolojisini kullanan bir ORM (object relational mapping – örneğin Hibernate) frameworkü olabilir, DAO interface sınıfı implemente edilir. UML diagramında yer aldığı gibi DummyDAOImpl ve JDBCDAOImpl isminde iki implementasyon sınıfı oluşturulmuştur. JDBCDAOImpl, JDBC API'sini kullanarak DAO işlemlerini gerçekleştirmektedir, yani verilere ulaşmak için JDBC API'de yer alan Connection, ResultSet, PreparedStatement gibi sınıflar kullanılmaktadır. DummyDAOImpl sınıfı ise, bilgibankasına gerek kalmadan, sistem testlerini kolaylaştırmak için oluşturulmuş sabit veriler sağlayan bir implementasyondur. Örneğin Hibernate ORM frameworkünü kullanarak HibernateDAOImpl isminde bir implementasyon sınıfı oluşturabiliriz. Bu implementasyon bünyesinde Hibernate teknolojisi kullanılarak bilgibankası işlemleri gerçekleştirilecektir. Sonuç itibariyle verilerin hangi teknolojiler kullanılarak elde edildiği DAO'nun kullanıldığı yerde önemini yitirmektedir. Bu şekilde katmanlar arası esnek bir mimari oluşturularak, bakımı ve geliştirilmesi kolay kod yazılımı yapılabilmektedir.

UML diagramında yer alan Test sınıfı gösterim katmanında yer alan bir JSP sayfası olarak düşünülebilir. Bu sınıfın görevi bilgibankasının **test_data** tablosunda yer alan verileri edinerek, ekranda görüntülemektir.

test_data tablosu aşağıdaki yapıya sahiptir:

id	test1	test2
1	Abc	def
2	123	345
3	xxx	yyy

Bu tablo anlam taşıyan veriler ihtiva etmese de, DAO tasarım şablonunun çalışma tarzını göstermek açısından yeterli olacaktır. **test_data** tablosunun her satırını (record) bir nesne bünyesinde tutabilmek için TestData isminde bir sınıf tanımlıyoruz:

```
package org.javatasarim.pattern.dao;

/**
 * TestData sinifi
 *
 * @author Oezcan Acar
 *
 */
public class TestData
{
      private String test1;
      private String test2;
      private int id;

      public int getId()
      {
```

```
            return id;
      }

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

      public String getTest1()
      {
            return test1;
      }

      public void setTest1(String test1)
      {
            this.test1 = test1;
      }

      public String getTest2()
      {
            return test2;
      }

      public void setTest2(String test2)
      {
            this.test2 = test2;
      }
}
```

TestData sınıfında, **test_data** tablosunun her kolonunda yer alan verileri tutabilmek için ***id***, ***test1*** ve ***test2*** isimlerinde değişkenler tanımlıyoruz. ***id*** int, ***test1*** ve ***test2*** String veri tipindendir. Test sınıfı ApplicationController sınıfından sadece TestData nesnelerini alacak ve bu nesnelerin sahip olduğu değerleri ekranda görüntüleyecektir.

Test sınıfı aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.dao;

import java.util.List;


/**
 * DAO tasarım sablonu için
 * olusturulan Test sinifi.
 *
 * @author Oezcan Acar
 *
 */
public class Test
{

      /**
       * Iki değişik DAO implementasyon sinifini
       * test eder.
       * @param args
       */
      public static void main(String[] args)
      {
```

```
            // JDBCDAOImpl implementasyonunun
            // calisabilmesi için bilgisayarinizda
            // bir mysql serverinin calismasi gerekmektedir.
            // Detaylar için JDBCDAOImpl sinifina bakiniz.
            ApplionController controller =
                  new ApplicationController(new JDBCDAOImpl());
            List<TestData> list =  controller.getTestDataList();

            for(int i=0; i < list.size(); i++)
            {
                  TestData data = list.get(i);
                  System.out.println(data.getTest1());
                  System.out.println(data.getTest2());
            }

            controller.setDao(new DummyDAOImpl());

            list =  controller.getTestDataList();

            for(int i=0; i < list.size(); i++)
            {
                  TestData data = list.get(i);
                  System.out.println(data.getTest1());
                  System.out.println(data.getTest2());
            }
      }
}
```

Test sınıfı, ApplicationController sınıfı üzerinden tüm TestData nesnelerini ihtiva eden bir liste edinir. Test sınıfı kendi başına JDBC yada başka bir teknoloji kullanarak bilgibankasına bağlanıp, verileri edinmiyor. Bu şekilde olması, Test sınıfını JDBC ya da kullanılan başka bir API[2]'ye bağımlı kılardı ve değiştirilmesi ve bakımı çok zorlaşırdı. Bir ApplicationController nesnesi oluşturmak için konstrüktör (constructor) parametresi olarak bir DAO interface implementasyon sınıfı kullanmamız gerekiyor. DAO interface sınıfı aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.dao;

import java.util.List;

/**
 * DAO tasarım sablonu için bir interface
 * sinif tanimliyoruz.
 *
 * @author Oezcan Acar
 *
 */
public interface DAO
{
      /**
       * Bilgibankasından testdata listesini
       * almak için tanimlanan metod.
       * @return List<TestData> testdata listesi
       */
      List<TestData> getTestDataList();

      /**
       * ID kolon degeri verilen bir
```

[2] Bakınız: http://en.wikipedia.org/wiki/API

```
         * testdata recordunu bulmak
         * için kullanılan motod.
         * @param id TestData record id
         * @return TestData bulunan testdata
         */
        TestData getTestData(int id);
}
```

DAO interface sınıfı getTestDataList() ve getTestData() isimlerinde iki metot tanımlar. ApplicationController sınıfı sadece bu interface sınıfını kullanacak şekilde programlanır. Nasıl olduğunu daha sonra göreceğiz.

Test.main() metodunun ilk bölümünde JDBCDAOImpl ikinci bölümünde DummyDAOImpl sınıflarını kullanarak ApplicationController nesneleri üretiyoruz. ApplicationController sınıfı aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.dao;

import java.util.List;

/**
 * DAO tasarım sablonunu
 * kullanan Controller sinifi.
 *
 * @author Oezcan Acar
 *
 */
public class ApplicationController
{
	private DAO dao;

	public ApplicationController(DAO dao)
	{
		this.dao = dao;
	}

	public DAO getDao()
	{
		return dao;
	}

	public void setDao(DAO dao)
	{
		this.dao = dao;
	}

	/**
	 * DAO üzerinden bir TestData nesnesi
	 * edinir.
	 *
	 * @param id int
	 * @return TestData
	 */
	public TestData getTestData(int id)
	{
		return dao.getTestData(id);
	}
```

```
	/**
	 * DAO üzerinden tüm TestData listesini
	 * edinir.
	 * @return List TestData listesi
	 */
	public List<TestData> getTestDataList()
	{
		return dao.getTestDataList();
	}

}
```

ApplicationController sınıfında dao isminde ve DAO tipinde bir sınıf değişkeni tanımlıyoruz. Test sınıfı tarafından controller.getTestData() ve controller.getTestDataList() metodları kullanıldığında, controller sınıfı bünyesinde bu istekler DAO.getTestData() ve DAO.getTestDataList() metodlarına delege (delegation) edilecektir. DAO interface sınıfını kullanarak değişik implementasyon sınıfları oluşturduğumuz taktirde, Test sınıfının veri edinme tarzını istediğimiz şekilde değiştirme imkanına sahip olabiliriz. Yapmamız gereken tek şey bir ApplicationController nesnesi oluştururken, sınıf konstrüktörüne istediğimiz DAO implementasyon sınıfını belirmek olacaktır:

```
ApplicationController controller =
                  new ApplicationController(new JDBCDAOImpl());
List<TestData> list =  controller.getTestDataList();
```

Yukarda yer alan satırda dolaylı olarak (controller getTestDataList() üzerinden) JDBCDAOImpl sınıfının getTestDataList() metodu işlem görür. JDBCDAOImpl sınıfı DAO sınıfını implemente eder ve aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.dao;

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

/**
 * DAO interface sinifini implemente
 * eden altsinif. JDBC teknolojisini
 * kullanarak DAO interface sinifini
 * implemente eder.
 *
 * @author Oezcan Acar
 *
 */
public class JDBCDAOImpl implements DAO
{

	/**
	 * JDBC Mysql URL
	 */
	private String url = "JDBC:msql://localhost:4333/" +
```

```
                "test_db";


    /**
     * ID kolon degeri verilen bir
     * testdata recordunu bulmak
     * için kullanılan motod.
     * JDBC kullanir.
     * @param id TestData record id
     * @return TestData bulunan testdata
     */
    public TestData getTestData(int id)
    {
          Connection con = null;
          PreparedStatement pstmt = null;
          ResultSet rs = null;
          TestData data = null;
          try
          {
                con = getConnection();
                pstmt = con.prepareStatement("select test1, test2, id" +
                            " from test_db where id=?");
                pstmt.setInt(1, id);
                rs = pstmt.executeQuery();
                data = new TestData();
                if(rs.next())
                {
                      data.setTest1(rs.getString(1));
                      data.setTest1(rs.getString(2));
                      data.setId(rs.getInt(3));
                }
                rs.close();
                pstmt.close();
          }
          catch (Exception e)
          {
                e.printStackTrace();
                throw new RuntimeException(e);
          }
          finally
          {
                if(con != null)
                {
                      try
                      {
                            con.close();
                      }
                      catch (SQLException e)
                      {
                            throw new RuntimeException(e);
                      }
                }
          }
          return data;
    }



    /**
     * Bilgibankasından testdata listesini
     * almak için tanimlanan metod.
```

```
	 * JDBC kullanir.
	 * @return List<TestData> testdata listesi
	 */
	public List<TestData> getTestDataList()
	{
		Connection con = null;
		ArrayList<TestData> list = new ArrayList<TestData>();
		PreparedStatement pstmt = null;
		ResultSet rs = null;
		try
		{
			con = getConnection();
			pstmt = con.prepareStatement("select test1, test2, id" +
					" from test_db");
			rs = pstmt.executeQuery();

			while(rs.next())
			{
				TestData data = new TestData();
				data.setTest1(rs.getString(1));
				data.setTest1(rs.getString(2));
				data.setId(rs.getInt(3));
				list.add(data);
			}
			rs.close();
			pstmt.close();
		}
		catch (Exception e)
		{
			throw new RuntimeException(e);
		}
		finally
		{
			if(con != null)
			{
				try
				{
					con.close();
				}
				catch (SQLException e)
				{
					throw new RuntimeException(e);
				}
			}
		}
		return list;
	}


	/**
	 * Bilgibankasına baglanmak için gerekli
	 * Connection nesnesini olusturur.
	 * @return Connection db bağlantısı
	 */
	public Connection getConnection()
	{
		Connection con = null;
		try
		{
			Class.forName("com.mysql.JDBC.Driver");
			con = DriverManager.getConnection(url, "test", "test");
```

```
            }
            catch (Exception e)
            {
                  e.printStackTrace();
                  throw new RuntimeException(e);
            }
            return con;
      }

}
```

JDBCDAOImpl sınıfı isminden de belli olduğu gibi JDBC teknolojisini kullanarak verilere ulaşmaktadır.

```
controller.setDao(new DummyDAOImpl());
list =  controller.getTestDataList();
```

Yukarda yer alan satırda dolaylı olarak (controller getTestDataList() üzerinden) DummyDAOImpl sınıfının getTestDataList() metodu işlem görür. DummyDAOImpl sınıfı DAO sınıfını implemente eder ve aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.dao;

import java.util.ArrayList;
import java.util.List;

/**
 * DAO interface sinifi için
 * dummy implementasyon örnegi.
 *
 * @author Oezcan Acar
 *
 */
public class DummyDAOImpl implements DAO
{

      /**
       * Bir dummy TestData olusturarak
       * geri verir. JDBC Örneğinde
       * bu metod bilgibankasına baglanarak
       * edindigi veriler ile bir TestData
       * olusturur.
       */
      public TestData getTestData(int id)
      {
            TestData data = new TestData();
            data.setTest1("test1");
            data.setTest2("test2");
            return data;
      }


      /**
       * Dummy TestData nesneleri olusturarak
       * bir ArrayList içinde geri verir.
       * JDBC Örneğinde bu metod bilgibankasına
       * baglanarak edindigi verilerle TestData
       * nesneleri olusturur ve bir ArrayList
```

```
         * içinde geri verir.
         */
        public List<TestData> getTestDataList()
        {
                ArrayList<TestData> list = new ArrayList<TestData>();

                TestData data = new TestData();
                data.setTest1("test1");
                data.setTest2("test2");

                list.add(data);

                data = new TestData();
                data.setTest1("test11");
                data.setTest2("test22");

                list.add(data);

                return list;
        }
}
```

DummyDAOImpl bünyesinde TestData nesneleri bilgibankasından elde edilen verilerle değil, sabit değerler kullanılarak oluşturulur. Örneğin DummyDAOImpl sınıfı JUnit[3] testleri bünyesinde DAO implementasyon sınıfı olarak kullanılabilir. DummyDAOImpl implementasyonu bilgibankasına ihtiyaç duymadığı için, gerekli TestData nesneleri kolay bir şekilde oluşturulup, diğer katmanlar tarafından kullanılabilir.

Örnekte görüldüğü gibi Test ve ApplicationController sınıfları etkilenmeden DAO sınıf implementasyonunu değiştirerek, veri edinme yöntemlerini değiştirebiliyoruz. DAO tasarım şablonunun kullanılma amacı işte budur! DAO kullanılarak değişik tarzda verilere ulaşım mekanizmaları implemente edilebilir. Diğer katmanlar etkilenmeden değişik DAO implementasyonları kullanılabilir.

DAO tasarım şablonu ne zaman kullanılır?

- Bilgibankası yada başka bir veritabanında yer alan verilere ulaşmak için DAO kullanılır.
- Verilere ulaşmayı sağlayan katman ile verileri kullanan katmanlar arasındaki bağımlılığı azaltmak ve bu katmanların hangi veritabanı tipinin kullanıldığını bilmesine gerek kalmadan işlem yapabilmelerini kolaylaştırmak için DAO kullanılır.
- Değişik tipte veritabanı sistemlerine (örneğin LDAP[4],XML, RDBMS[5]) aynı metotları kullanarak erişimi standart hale getirmek için DAO kullanılır. Böylece veritabanı operasyonları için kullanılan metotlar standardize edilmiş olur ve tüm sistem içinde tekil olarak kullanılabilir.

İlişkili tasarım şablonları:

[3] Bakınız: http://www.junit.org
[4] Bakınız: http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
[5] Bakınız: http://en.wikipedia.org/wiki/RDBMS

- DAO implementasyon sınıfları DTO (Data Transfer Objects) tasarım şablonunu kullanarak verilerin diğer katmanlar tarafından kullanılmasını sağlarlar.
- Factory tasarım şablonu kullanılarak istenilen tipte DAO implementasyon sınıfları oluşturulabilir.