# Test Güdümlü Yazılımın Tasarım Üzerindeki Etkileri

## KurumsalJava.com

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

Yazılımcı olarak çalıştığım projelerde geleneksel[1] ve çevik yazılım süreçleri[2] hakkında tecrübe edinme firsatı buldum. En son kitabım bir çevik süreç olan Extreme Programming[3] hakkındadır. Edindiğim tecrübeler doğrultusunda çevik süreçlerin, klasik yazılım süreçlerine nazaran bakımı ve geliştirilmesi daha kolay yazılım sistemlerinin oluşturulmasında daha avantajlı olduğunu söyleyebilirim.

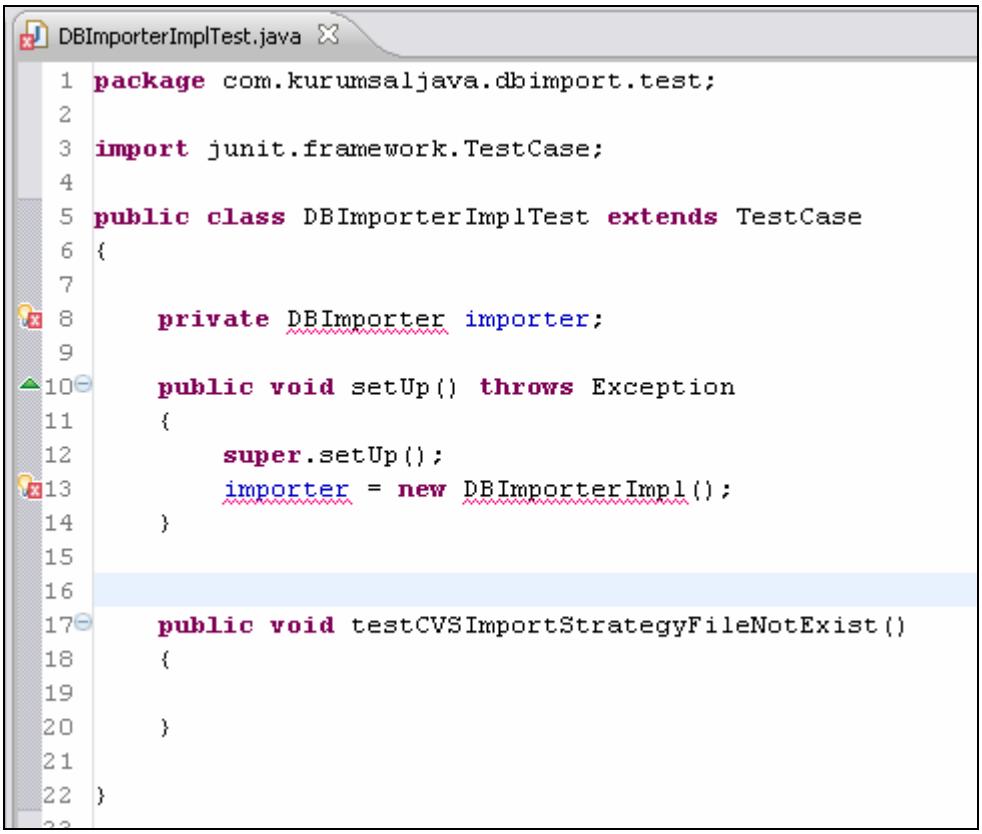
Bu yazımda sizelere test güdümlü yazılım sürecinin, yazılım tasarımı üzerindeki etkilerini bir örnek üzerinde aktarmak istiyorum. TDD[4] ile birlikte oluşan tasarım, kendiliğinden oluşan birşey değildir. Testler şekil aldıkça, oluşturmak istediğimiz tasarımın modeli de gözümüzde canlanmaya başlar. Oluşturduğumuz testler, programın gelecekteki kullanıcılarını (client) simule ettiği için, programın nasıl kullanılacağını testler bünyesinde gözlemlemek kolaylaşmaktadır. Bu süreç, sınıfların ve metotların kullanıcı gözüyle (client) tasarlanmasını sağlar. Bu sayede basit ve kullanışlı API (Application Programming Interface)'ler oluşur. Test güdümlü yazılım tasarımı devamlı zorlar ve yetersiz kaldığı yerlerde *refactoring* yöntemleriyle yenilenmesini sağlar. Bu süreç sayesinde kendisini devamlı yenileyen ve yeni gereksinimlere cevap veren bir tasarım oluşur.

Çalıştığım proje bünyesinde entegrasyon testlerini otomatize edebilmek için, bilgibankasında yer alması gereken verileri herhangi bir kaynaktan edinip, bilgibankasına aktaran bir program parcasına ihtiyaç duyulmaktadır. *DBImport*er ismini verdigim bu programı, TDD teknikleri kullanarak nasıl implemente edebileceğimizi şimdi hep beraber yakından inceleyelim. Testler ilerledikçe tasarımın nasıl oluştuğunu ve hangi kararların tasarımı şekillendirdiğini örnek üzerinde inceleme firsatı bulacağız.

İlk işlem olarak *DBImportImplTest* ismini taşıyan bir JUnit[5] test sınıfı oluşturuyoruz.

---

[1] Şelale (Waterfall) yazılım yöntemi hakkındaki yazımı http://www.kurumsaljava.com/2009/02/26/yazilimda-selale-waterfall-yontemi/ adresinden temin edebilirsiniz.
[2] Çevik süreç nedir başlıklı yazımı http://www.kurumsaljava.com/2008/12/02/cevik-surec-agile-process-nedir/ adresinden temin edebilirsiniz.
[3] http://www.kurumsaljava.com/2009/02/08/turkiyenin-ilk-extreme-programming-konulu-kitabi/
[4] TDD hakkındaki yazımı http://www.kurumsaljava.com/2008/11/26/test-gudumlu-yazilim-test-driven-development-tdd/ adresinden edinebilirsiniz.
[5] Junit hakkındaki yazımı http://www.kurumsaljava.com/2008/11/28/unit-testing-konseptleri/ adresinden temin edebilirsiniz.




**Resim 1**

*Resim 1* de görüldüğü gibi sekizinci satırda *DBImporter* tipinde bir sınıf değişkeni tanımlıyoruz. Bu test etmek istediğimiz program parcasıdır. Onuncu satırda *setUp()* metodunu tanımlıyoruz. Her test öncesi *setUp()* metodu test için gerekli alt yapının oluşmasında kullanılan bir metotdur.

*DBImporter* bir Java interface sınıfıdır. Bir interface sınıf kullanarak ilk tasarım kararımızı vermiş oluyoruz. Neden bir interface sınıfta karar kıldık? *DBImporter* programını bir komponent olarak düşünüyorum. Bir komponentin belirli iletişim noktaları vardır. Bunlar bir interface sınıfında tanımlanmış metotlardır. Bunun haricinde kullanıcı sınıflar, bir komponent bünyesinde olup, bitenlerden habersizlerdir. Bu kullanıcı ve kullanılan arasında esnek bir bağın[6] oluşmasını sağlar. Esnek bağımlılıklardan oluşan bir yazılım sisteminin bakımı ve geliştirilmesi çok daha kolaydır. Bu sebepten dolayı *DBImporter* programını bir komponent olarak tasarlamak en uygun seçim olacaktır.

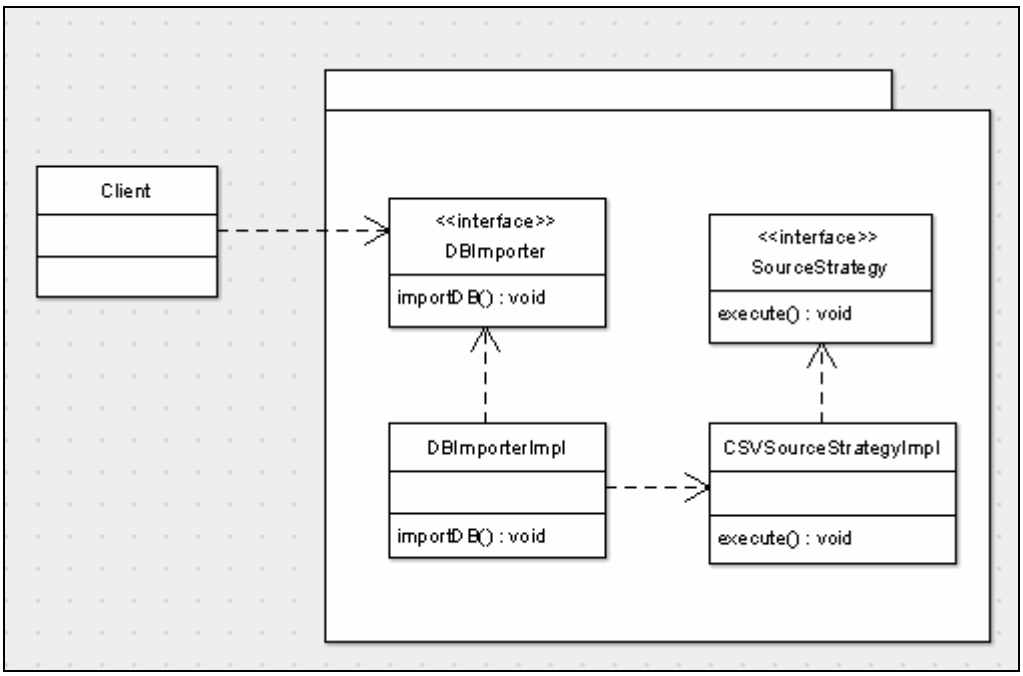
*Resim 1*, on yedinci satırda ilk JUnit test sınıfını tanımlamış olduk. Seçmiş olduğumuz test ismi, bizi ikinci tasarım kararına doğru götürüyor gibi…. Devam edelim ve görelim.

JUnit test metotlarında somut öğeleri test etmemiz gerekiyor. İlk JUnit metot isminden de anlaşılacağı gibi, bir CSV dosyasında bulunan verileri bilgibankasına aktarmak için kolları sıvıyoruz. Bir CSV dosyası bir .txt dosyadır. İhtiva ettiği veriler ; ile birbirinden ayrılır. *DBImporter* programı bir CSV dosyasındaki verileri edinebilip, bilgibankasına ekleyebilmelidir.

[6] Esnek bağ hakkındaki yazımı http://www.kurumsaljava.com/2009/10/16/loose-coupling-Ic-esnek-bag-tasarim-prensibi/ adresinden edinebilirsiniz.

Programcı olarak görevimiz, *DBImporter* programını, gelecekte oluşacak değişikliklere izin verecek şekilde tasarlamak ve implemente etmektir. Programa yeni özellikler ekleyerek, mevcut kodu degiştirmek zorunda kalmadan, genişletebilmemiz gerekmektedir. Tasarımdaki bu amacımıza *Open Closed* (açık kapalı) [7] tasarım prensibini uygulayarak ulaşabiliriz. Strateji[8] tasarım şablonunu kullanarak, OCP tasarım prensibini nasıl uygulayacağımızı yakından inceleyeceğiz.

*DBImporter* programının değişik kaynaklardan gelen verileri kullanabilmesi gerekmektedir. Bugün belki veri kaynağı bir CSV dosyası olabilir. Belki gelecekte veri kaynağı olarak başka bir bilgibankasını kullanmak zorunda kalabiliriz. Bu gibi değişiklikleri göz önünde bulundurmak için strateji tasarım şablonundan faydalanabiliriz.




**Resim 2**

*Resim 2* de kafamızda oluşan model yer almaktadır. *SourceStrategy* isminde, değişik veri kaynaklarını represente eden bir interface sınıf bulunmaktadır. Bu interface sınıfın ilk implementasyonu *CSVSourceStrategyImpl* sınıfıdır. Bu sınıf bünyesinde bir CSV dosyasında yer alan veriler bilgibankasına import edilecektir.

*Resim 2* de yer alan modelden yola çıkarak, ilk JUnit testini *kod 1* deki gibi şekillendiriyoruz.

[7] OCP (open closed principle) tasarım prensibi hakkındaki yazımı http://www.kurumsaljava.com/2009/10/16/open-closed-principle-ocp-acik-kapali-tasarim-prensibi/ adresinden temin edebilirsiniz.
[8] Strateji tasarım şablonu hakkındaki yazımı http://www.kurumsaljava.com/2008/12/15/strategy-strateji-tasarim-sablonu/ adresinden termin edebilirsiniz.

```java
package com.kurumsaljava.dbimport.test;

import java.io.File;
import junit.framework.TestCase;
import com.kurumsaljava.dbimport.CVSSourceStrategy;
import com.kurumsaljava.dbimport.DBImporterImpl;
import com.kurumsaljava.dbimport.SourceStrategy;


public class DBImporterImplTest extends TestCase
{

	private DBImporterImpl importer;
	private SourceStrategy sourceStrategy;

	public void setUp() throws Exception
	{
		super.setUp();
		importer = new DBImporterImpl();
	}


	public void testCSVImportFileNotFound()
	{
		File file = new File("test.cvs");
		sourceStrategy = new CVSSourceStrategyImpl(file);

		try
		{
			sourceStrategy.execute();
		}
		catch (IllegalArgumentException e)
		{
			assertTrue(true);
		}
		catch (Exception e)
		{
			fail();
		}

	}
}
```

**Kod 1**

İlk JUnit testi kapsamında (*testCSVImportFileNotFound()*) yeni oluşturacağımız *CSVSourceStrategyImpl* implementasyonunun, gerekli CSV dosyası bulunamaması durumunda gösterdiği davranış biçimini test edeceğiz. Test güdümlü yazılımda testler, test edilen programın sahip olacağı tüm işlevlerin adım adım testlere dökülmesiyle oluşur. Testler tamamladığında, testlerle oluşan programın tüm işlevleri otomatik olarak test edilebilir. İlk oluşturduğumuz test, programın en basit işlevini test etmektedir: „CSV dosyası bulunamaması durumunda program nasıl bir davranış göstermektedir.?“ Test bünyesinde, programdan beklediğimiz davranış tarzını *assert* komutlarıyla kodda ifade ediyoruz. Test ettiğimiz implementasyon *kod 2* de yer almaktadır.

```
package com.kurumsaljava.dbimport;

import java.io.File;

public class CSVSourceStrategyImpl implements SourceStrategy {

	private File csv;

	public CVSSourceStrategyImpl(File file)
	{
		this.csv = file;
	}

	public void execute()
	{
		if (this.csv == null || !this.csv.exists())
			throw new IllegalArgumentException("cvs file not
found");

	}
}
```

**Kod 2**

Mevcut implementasyon (*kod 2*) testin olumlu bir şekilde çalışmasını sağlamaktadır. Şimdi yeni bir test oluşturarak, diğer işlevleri test edelim.

```
public void testCSVImportGetHeader()
{
	URL url = this.getClass().getResource("/com/kurumsaljava/" +
			"dbimport/test/test1.csv");
	File file = new File(url.getFile());
	sourceStrategy = new CSVSourceStrategyImpl(file);
	sourceStrategy.execute();
	assertTrue(((CSVSourceStrategyImpl) sourceStrategy)
			.getHeader().length == 3);
}
```

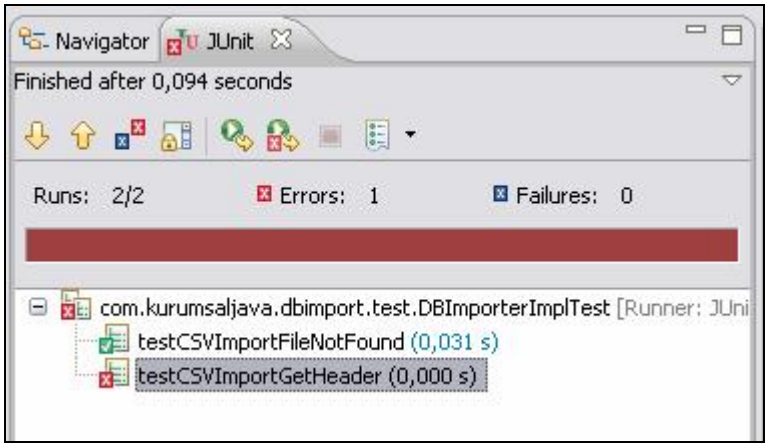
**Kod 3**

İmport işlemini gerçekleştirebilmek için, import yaptığımız bilgibankası tablosunun kolon isimlerini bilmemiz gerekmektedir. Bu amaçla CSV dosyasının ilk satırına kolon isimlerinden oluşan bir liste yerleştiriyoruz. CSV dosyasının ikinci satırından itibaren import edilecek veriler yer alacaktır. CSV dosyasının içeriği aşağıda yer almaktadır.

```
ISIM;SOYAD;DOGUMTARIHI
Özcan;Acar;1974
Ahmet;Yildirim;1955
Figen;Tas;1981
```

***test1.csv***

*Kod 3* de yer alan test ile, CSV dosyasının ilk satırında yer alan kolon isimlerinin elde edilisini test ediyoruz.




**Resim 3**

Bu testi çalıştırdığımız taktirde *resim 3* deki gibi bir netice alırız. Birinci test çalışırken, ikinci test olumsuz sonuç vermektedir. Bu doğaldır, çünkü henüz testi olumlu hale getirmek için gerekli kodu oluşturmadık.

*Kod 3* de yer alan testin ne ifade ettiğini, daha doğrusu programın hangi işlevini test ettiğini tekrar gözden geçirelim. İmport işlemini gerçekleştirebilmek için, bilgibankası tablosunun kolonlarını tanımamız gerekiyor. Test ettiğimiz *CSVSourceStrategyImpl* sınıfın CSV dosyasının ilk satırında yer alan kolon listesini edinip, kontrol amacıyla tekrar bize geri verebilmelidir. CSV dosyasından edindiğimiz kolon isimlerini ve verileri tutmak için *CSVSourceStrategyImpl* sınıfı bünyesinde *header* ve *line* isimlerinde iki değişken tanımlıyoruz. Test bünyesinde, kolon listesinin üç elementten oluştuğunu kontrol ediyoruz.

```
package com.kurumsaljava.dbimport;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;

public class CSVSourceStrategyImpl implements SourceStrategy {

	private File csv;
	private String[] header;
	private List<String[]> line;

	public CSVSourceStrategyImpl(File file)
	{
		this.csv = file;
	}

	public void execute()
	{
		if (this.csv == null || this.csv.exists() == false)
			throw new IllegalArgumentException("cvs not found");

		parseHeader();

	}
```

```
	private void parseHeader()
	{
		BufferedReader input = null;
		try
		{
			String line = null;
			input = new BufferedReader(new FileReader(this.csv));
			while ((line = input.readLine()) != null)
			{
				this.header = line.split("[;]");
				break;
			}
		}
		catch (Exception e)
		{
			throw new RuntimeException(e);
		}
		finally
		{
			try
			{
				input.close();
			}
			catch (IOException e)
			{
				throw new RuntimeException(e);
			}
		}
	}


	public File getCsv()
	{
		return csv;
	}

	public void setCsv(File csv)
	{
		this.csv = csv;
	}

	public String[] getHeader()
	{
		return header;
	}

	public void setHeader(String[] header)
	{
		this.header = header;
	}

	public List<String[]> getLine()
	{
		return line;
	}

	public void setLine(List<String[]> line)
	{
		this.line = line;
	}
}
```

**Kod 4**

*Kod 4* de yer alan implementasyon, *kod 3* de yer alan testin başarılı sonuç vermesi için gerekli kodu ihtiva etmektedir.

Bir sonraki testimizde, CSV dosyasında yer alan verilerin elde edilişini test edelim. Yeni test metodu *kod 5* de yer almaktadır.

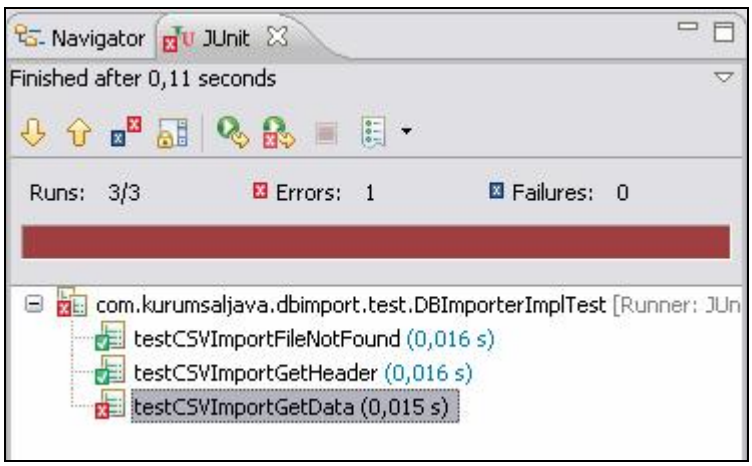
```
public void testCSVImportGetData()
{
      URL url = this.getClass().getResource("/com/kurumsaljava/" +
                  "dbimport/test/test1.csv");
      File file = new File(url.getFile());
      sourceStrategy = new CSVSourceStrategyImpl(file);
      sourceStrategy.execute();
      assertTrue(((CSVSourceStrategyImpl) sourceStrategy)
                  .getLine().size() == 3);
}
```

**Kod 5**

```
ISIM;SOYAD;DOGUMTARIHI
Özcan;Acar;1974
Ahmet;Yildirim;1955
Figen;Tas;1981
```

***test1.csv***

Testlerde kullandığımız *test1.csv* dosyasını yakından incelediğimizde, üç satırlık veri ihtiva ettiğini görmekteyiz. *Kod 5* de yer alan test, bu üç satırlık veriyi test etmektedir.




**Resim 4**

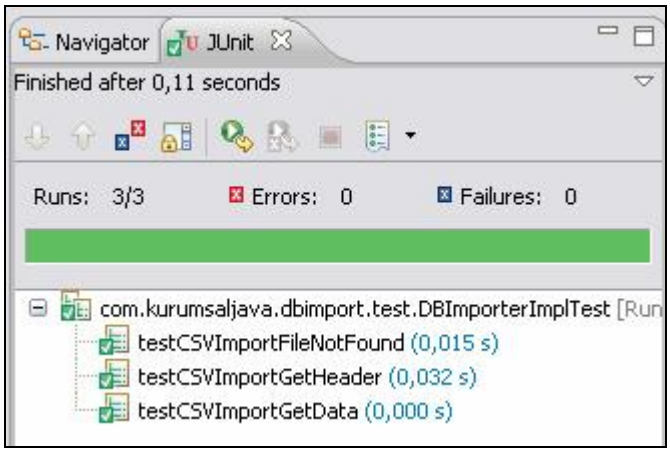
*Resim 4* de yer aldığı gibi en son oluşturduğumuz test olumlu sonuç vermemektedir, çünkü henüz bu testin olumlu sonuç vermesini sağlayacak kod implemente edilmemiştir **J**
*Kod 6* gerekli implementasyonu ihtiva etmektedir.

```
package com.kurumsaljava.dbimport;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class CSVSourceStrategyImpl implements SourceStrategy {

	private File csv;
	private String[] header;
	private List<String[]> line;

	public CSVSourceStrategyImpl(File file)
	{
		this.csv = file;
	}

	public void execute()
	{
		if (this.csv == null || this.csv.exists() == false)
			throw new IllegalArgumentException("cvs not found");

		parseHeader();
		parseData();

	}

	private void parseData()
	{
		BufferedReader input = null;
		this.line = new ArrayList<String[]>();
		try
		{
			String line = null;
			input = new BufferedReader(new FileReader(this.csv));
			int counter = 0;
			while ((line = input.readLine()) != null)
			{
				if(counter != 0)
				{
					String[] temp = line.split("[;]");
					this.line.add(temp);
				}
				counter++;
			}
		}
		catch (Exception e)
		{
			throw new RuntimeException(e);
		}
		finally
		{
```

```
                try
                {
                        input.close();
                }
                catch (IOException e)
                {
                        throw new RuntimeException(e);
                }
        }

    }

    private void parseHeader()
    {
        BufferedReader input = null;
        try
        {
                String line = null;
                input = new BufferedReader(new FileReader(this.csv));
                while ((line = input.readLine()) != null)
                {
                        this.header = line.split("[;]");
                        break;
                }
        }
        catch (Exception e)
        {
                throw new RuntimeException(e);
        }
        finally
        {
                try
                {
                        input.close();
                }
                catch (IOException e)
                {
                        throw new RuntimeException(e);
                }
        }
    }


    public File getCsv()
    {
        return csv;
    }

    public void setCsv(File csv)
    {
        this.csv = csv;
    }

    public String[] getHeader()
    {
        return header;
    }

    public void setHeader(String[] header)
    {
        this.header = header;
    }
```

```
	public List<String[]> getLine()
	{
		return line;
	}

	public void setLine(List<String[]> line)
	{
		this.line = line;
	}
}
```

**Kod 6**




**Resim 5**

Oluşturmuş olduğumuz ilk üç test bir CSV dosyasından kolon isimlerini ve verileri edinebilen bir implementasyonun oluşmasını sağladı. Bunun yanısıra testler tasarımın şekillenmesinde katkıda bulundular. Kullandığımız tasarım elementleri şu şekildedir:

- Bir **interface** sınıf kullanarak *DBImporter* programını bir komponent olarak tasarladık. Bir interface sınıf kullanarak, komponent ve kullanıcıları arasında bir nevi anlaşma metni tanımladık. Kullanıcı sınıflar, interface sınıfında tanımlanmış olan metotlar aracılığı ile, komponentin dış dünyaya sunduğu hizmetlerden faydalanabilirler. Kullanıcı sınıflar komponent tarafından sunulan hizmetlerin ne şekilde implemente edildiklerini bilmek zorunda değildirler. Bu şekilde kullanıcı sınıflar ve komponentler arasında *esnek bir bağ* oluşturmuş oluyoruz. Esnek bağ sayesinde dış dünyayı etkilemeden komponent bünyesinde istediğimiz değişikliği gerçekleştirebiliriz. Bu şekilde bir yol almamız, oluşturduğumuz yazılım sisteminin kırılganlık oranını düşürmektedir.
- **Open Closed** prensibini uygulayabilmek için **strateji tasarım şablonunu** seçtik. İlk strateji implementasyonu CSV dosyalarından edinilen verilerin bilgibankasına import edilme işlemi için gerçekleştirdik.
- Interface sınıflar kullanarak bağımlılıkların tersine çevrilmesi[9] (Dependency Inversion Principle – **DIP**) tasarım prensibini de uygulamış olduk.

[9] DIP hakkındaki yazımı http://www.kurumsaljava.com/2009/10/29/dependency-inversion-principle-dip-bagimliliklarin-tersine-cevrilmesi-prensibi/ adresinden termin edebilirsiniz.

İmport işlemini gerçekleştirebilmek için program tarafından SQL insert komutlarının oluşturulması gerekmektedir. Bu işlemi yapacak kodu oluşturmadan önce bu işlevi kontrol eden JUnit testini oluşturuyoruz.

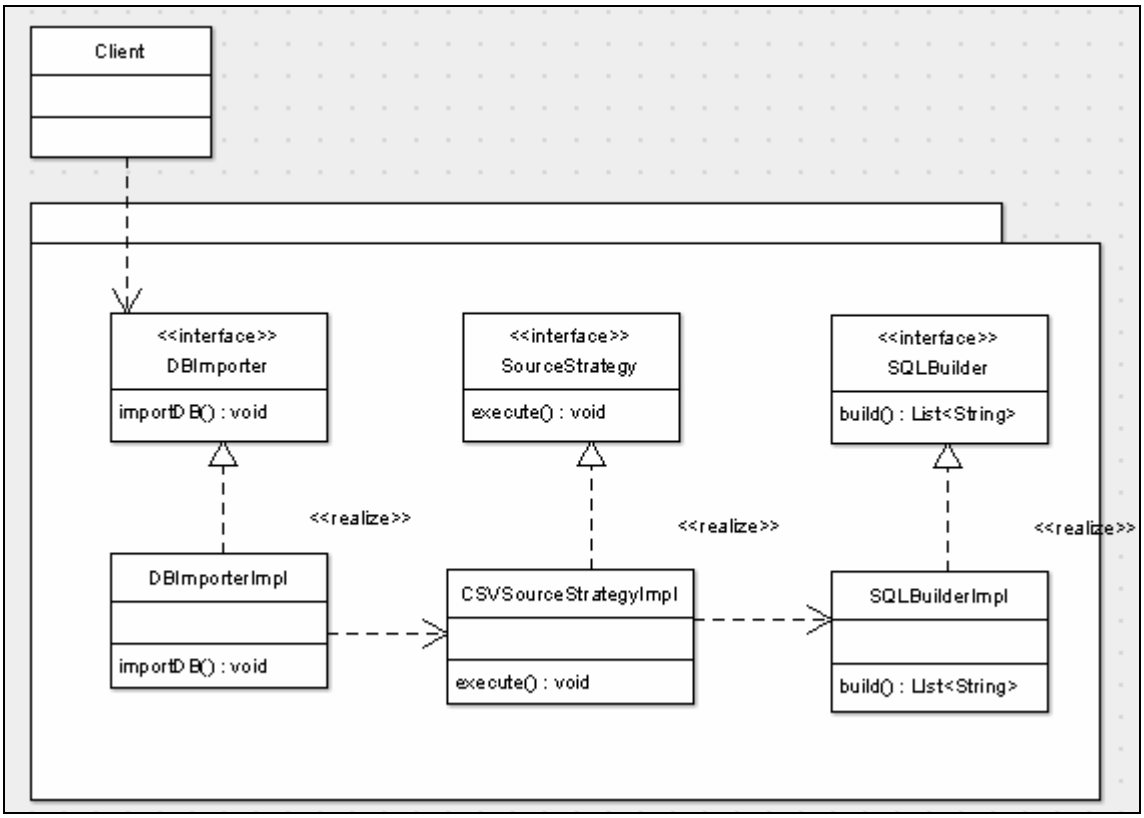
```
public void testCSVImportGetSQL()
{
      URL url = this.getClass().getResource("/com/kurumsaljava/" +
                  "dbimport/test/test1.csv");
      File file = new File(url.getFile());
      sourceStrategy = new CSVSourceStrategyImpl(file);
      sourceStrategy.execute();
      assertTrue(((CSVSourceStrategyImpl) sourceStrategy)
                  .getSql().size() == 6);
}
```

**Kod 7**

*Kod 7* de yer alan test, *test1.csv* dosyasında yer alan veriler kullanıldığında altı adet SQL komutunun *DBImporter* tarafından oluşturulması gerektiğini ifade etmektedir. *test1.csv* dosyası dört satırdan oluşmaktadır. Birinci satırda, kullanılan kolon listesi yer almaktadır. İkinci satırdan itibaren import edilecek veriler yer almaktadır. İmport edilecek toplam üç satır bulunmaktadır. SQL insert komutuyla verileri bilgibankasına eklemeden önce, SQL delete ile mevcut verileri bilgibankasından silmemiz gerekmektedir, aksi taktirde insert komutları, veri bilgibankasında mevcut olduğu için işlem görmeyebilir. Bunu engellemek için her insert komutu öncesi aynı veriyi bilgibankasından silen bir delete komutu koşturmamız gerekmektedir. Toplamda program tarafından oluşturulması gereken SQL komut adedi altıdır. Oluşturulması gereken SQL komutları bir sonraki tabloda yer almaktadır.

```
delete from customer where ISIM='Özcan' AND SOYAD='Acar' AND
DOGUMTARIHI='1974'
insert into customer(ISIM,SOYAD,DOGUMTARIHI,) values
('Özcan','Acar','1974')
delete from customer where ISIM='Ahmet' AND SOYAD='Yildirim' AND
DOGUMTARIHI='1955'
insert into customer(ISIM,SOYAD,DOGUMTARIHI,) values
('Ahmet','Yildirim','1955')
delete from customer where ISIM='Figen'AND SOYAD='Tas' AND
DOGUMTARIHI='1981'
insert into customer(ISIM,SOYAD,DOGUMTARIHI,) values
('Figen','Tas','1981')
```

**SQL 1**




**Resim 6**

*CSVSourceStrategyImpl* sınıfı gerekli SQL komutlarını kendisi oluşturmamalı, bunu görevi sadece SQL komutlarını oluşturmak olan başka bir sınıfa devretmelidir. Böyle bir tasarım kararı vererek, *CSVSourceStrategyImpl* sınıfının sorumluluk alanını **genişletmiyoruz**. Bu tek sorumluluk prensibinin [10] uygulanış biçimidir. Her sınıfın sadece bir sorumluluğu olmalıdır. Sınıfların sorumlulukları arttıkça, değişikliğe uğrama rizikoları artar. Bu durum yazılım sisteminin kırılganlık oranını artırır.

Gerekli SQL komutlarını oluşturabilmek için modelimizi *resim 6* da yer aldığı gibi genişletiyoruz. SQL komutlarını oluşturmak için *SQLBuilder* isminde bir interface sınıf oluşturuyoruz. Bu sınıf ve implementasyonu aşağıda yer almaktadır.

```
package com.kurumsaljava.dbimport;

import java.util.List;

public interface SQLBuilder
{
      List<String> build(String table, String[] header, List<String[]>
line);

}
```

**Kod 8**

[10] Tek sorumluluk prensibi hakkındaki yazımı http://www.kurumsaljava.com/2009/10/14/single-responsibility-principle-srp-tek-sorumluk-prensibi/ adresinden temin edebilirsiniz.

```java
package com.kurumsaljava.dbimport;

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

public class SQLBuilderImpl implements SQLBuilder
{

	public List<String> build(String table, String[] header,
List<String[]> line)
	{
		// precondition check
		if(table == null) throw new IllegalArgumentException("Table
not found");

		List<String> sql = new ArrayList<String>();

		StringBuilder sqlDelete = new StringBuilder();
		StringBuilder sqlInsert = new StringBuilder();

		StringBuilder headertemp = new StringBuilder();
		headertemp.append("(");

		for(int i=0; i < header.length; i++)
		{
			headertemp.append(header[i]);
			if(i-header.length != 1)
			{
				headertemp.append(",");
			}
		}
		headertemp.append(")");

		for(int i=0; i < line.size(); i++)
		{
			sqlDelete.setLength(0);
			sqlInsert.setLength(0);

			String[] temp = line.get(i);

			sqlDelete
				.append("delete from ")
				.append(table)
				.append(" where ");

			for(int x=0; x < header.length; x++)
			{
				sqlDelete
					.append(header[x])
					.append("=")
					.append("'")
					.append(temp[x])
					.append("'");

				if( (header.length-x) != 1)
				{
					sqlDelete
						.append(", ");
				}
			}

			sqlInsert
```

```
                        .append("insert into ")
                        .append(table)
                        .append(headertemp.toString())
                        .append(" ")
                        .append("values (");

                for(int x=0; x < temp.length; x++ )
                {
                      sqlInsert
                            .append("'")
                            .append(temp[x])
                            .append("'");

                      if((temp.length-x) != 1)
                      {
                            sqlInsert
                                  .append(",");
                      }

                }


                sqlInsert
                      .append(")");

                System.out.println(sqlDelete.toString());
                System.out.println(sqlInsert.toString());


                sql.add(sqlDelete.toString());
                sql.add(sqlInsert.toString());

            }
            return sql;
      }
}
```

**Kod 9**

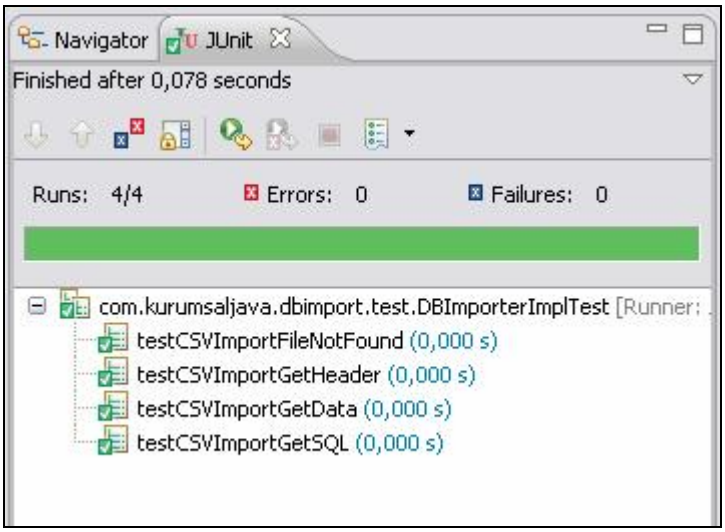
```
package com.kurumsaljava.dbimport;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class CSVSourceStrategyImpl implements SourceStrategy {

      private File csv;
      private String[] header;
      private List<String[]> line;
      private List<String> sql;


      public CSVSourceStrategyImpl(File file)
      {
            this.csv = file;
      }

      public void execute()
```

```
	{
		if (this.csv == null || this.csv.exists() == false)
			throw new IllegalArgumentException("cvs not found");

		parseHeader();
		parseData();
		buildSQL();

	}

	private void buildSQL()
	{
		SQLBuilder builder = new SQLBuilderImpl();
		setSql(builder.build("customer", this.header, this.line));

	}

	private void parseData()
	{
		BufferedReader input = null;
		this.line = new ArrayList<String[]>();
		try
		{
			String line = null;
			input = new BufferedReader(new FileReader(this.csv));
			int counter = 0;
			while ((line = input.readLine()) != null)
			{
				if(counter != 0)
				{
					String[] temp = line.split("[;]");
					this.line.add(temp);
				}
				counter++;
			}
		}
		catch (Exception e)
		{
			throw new RuntimeException(e);
		}
		finally
		{
			try
			{
				input.close();
			}
			catch (IOException e)
			{
				throw new RuntimeException(e);
			}
		}

	}

	private void parseHeader()
	{
		BufferedReader input = null;
		try
		{
			String line = null;
			input = new BufferedReader(new FileReader(this.csv));
			while ((line = input.readLine()) != null)
```

```
                {
                        this.header = line.split("[;]");
                        break;
                }
        }
        catch (Exception e)
        {
                throw new RuntimeException(e);
        }
        finally
        {
                try
                {
                        input.close();
                }
                catch (IOException e)
                {
                        throw new RuntimeException(e);
                }
        }
}


public File getCsv()
{
        return csv;
}

public void setCsv(File csv)
{
        this.csv = csv;
}

public String[] getHeader()
{
        return header;
}

public void setHeader(String[] header)
{
        this.header = header;
}

public List<String[]> getLine()
{
        return line;
}

public void setLine(List<String[]> line)
{
        this.line = line;
}

public List<String> getSql()
{
        return sql;
}

public void setSql(List<String> sql)
{
        this.sql = sql;
}
```

```
}
```

**Kod 10**




**Resim 7**

*Kod 8-10* da yer alan implementasyon ile *resim 7* de görüldügü gibi oluşturduğumuz en son test olumlu sonuç vermektedir.

*DBImporter* implementasyonunu tamamlamak için yeni testlere gerek vardır. Bu testler diğer testlere analog olarak geliştirilebilir. Ben burada yazima son noktayı koymadan önce, tekrar kısa bir özet oluşturmak istiyorum:

Görüldügü gibi testlerden yola çıkarak tasarımı oluşturmak ve yeni gereksinimler doğrultusunda adapte etmek mümkündür. Geleneksel yazılım süreçlerinde ne yazık ki tasarım, yazılım öncesinde en son detayına kadar oluşturulduğu için, bu tasarımın yazılım sistemi büyüdükçe, bu ağırlığı kaldırması zorlaşmaktadır. Test güdümlü oluşturulan yazılım sistemlerinde tasarım, implementasyona paralel olarak gelişir. Her yeni test sistemi genişlettiği için, mevcut tasarım sorgulanmış olur. Eğer tasarım implemente edilmek istenen gereksinim için yetersiz ise, *refactoring* yöntemleri kullanılarak istenilen tasarım uygulanabilir. *Refactoring* sadece otomatik olarak çalışabilen test kümesi mevcut ise yapılabilecek bir işlemdir. Aksi taktirde *refactoring* işleminden sonra oluşabilecek yan etkilerin tespiti çok zaman alıcı ve güç bir işlem haline dönüşebilir.

Test güdümlü yazılım tasarımı doğrudan etkiler ve kendi gereksinimleri doğrultusunda şekillendirir. Test güdümlü yazılım **uygulanmadığı** taktirde oluşacak program parçalarının (sınıf) yapısı programcının inisiyatifindedir. Programcı implementasyonu gerçekleştirirken, gereğinden daha fazlasını kodlama eğilimi gösterebilir. Bunun en büyük sebebi, programcının implementasyonu kullanıcı (client) gözüyle görmemesidir. Bu gereğınden fazlası ya da yanlış bir tasarımın olusmasını sağlayabilir. Programcının test güdümlü çalışması durumunda, oluşan kod sadece test edilen kod kadardır. Bunun yanısıra testler kullanıcı rolünde olduklarından, oluşan metotlar ve sınıflar çok sade yapıdadır ve gereksiz parametrelerle

yüklenmemişlerdir. Bu şekilde sade ama gelecekteki değişikliklere cevap verebilecek bir tasarım oluşur.

*EOF (End Of Fun)*
*Özcan Acar*