Alistair Cockburn
Kahve Makinesi Tasarımı

Tasarım nedir?

Bir tasarımı başka bir insana nasıl anlatırsın?

Burada bir örnek üzerinden göreceğiz, OO tasarımının en temel özelliklerini. Gerçekçi bir problem inceleyeceğiz, koşulların sürekli değiştiği bir problem.

Gündelik hayatta karşılaştığımız bir sistemin tasarımını çizin. Bir tane kağıt kullanın. Ancak hiçbir konuşmayla açıklama yapmayacaksınız. Mesela bir banka şubesinin tasarımını yapın. Ancak bütün sistemin olacak sadece bilgi sisteminin değil. 15 dakikanız var. Ve bunun sonucunda çıkardığınız taslak, sokaktan geçen birinin anlayabileceği bir şey olmalı. Haydi...

Bu soruya karşılık yapılan iyi tasarımlarda dört tane özellik görünüyor:

1. Kilit unsurların isimleri
2. Her unsurun amacı
3. Unsurların arasındaki önemli etkileşimler
4. Sunulan hizmetler

Bu dört şey bir tasarımın anlatılmasını sağlamakta çok etkili.

Bunların arasında en önemli şey unsurların sorumluluğu, yani amaçlarıdır.

Kalıtım da bazen problemleri çözmek için kullanılır. Ancak temel tasarım önemli olandır.

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\12.gif

Kahve Makinesi

Müşterimiz, basit bir kahve makinesi istiyor. Makine içine 35 cent atıldığında kahve verecek. Kahveye şeker ve krema eklenmesi seçenekleri var. Makineyi kısa zamanda teslim etmemiz bekleniyor.

Makinede bir tane bozuk para deliği, bozuk para dönüş kutusu, üstünü geri verme kolu ve dört tane düğme olacak: kremalı, kremasız, kremalı ve şekerli, kremasız ve şekerli.

Şimdi makineyi önceki örnekte gördüğümüz gibi tasarlayın...

Çözüm (En doğru çözüm değil, daha iyisini yapmış olabilirsiniz):

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\13.gif

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\14.gif

Et suyu

Bir süre sonra yeni bir ihtiyaç ekleniyor. Et suyu çorbası da verilecek. Fiyatı 25 cent olacak.

Et suyu için bir düğme ve et suyu tozu için de bir kap koyuyoruz. Tasarımdaki değişiklikler nasıl olacak?

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\15.gif

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\16.gif

Durum analizi:

Nesne odaklılık değişimin etkisini azaltıyor deniyordu. Halbuki mevcut tasarımı değiştirdik.

Temel hata, cash boxın kahvenin fiyatını bilmesiydi.

Şimdi front panel hem seçimi hem de her bir seçimin fiyatını biliyor. Fiyatların değişmesi sadece onu etkiler.

Ancak burada da bir sorun var. Front panel çok şey biliyor.

Kimlik kartı okuyucu

Müşteri kimlik kartı okuyucusu kullanan firmaları görüyor. Kahvenin masrafının doğrudan doğruya çalışanın maaşından düşülmesini öneriyor. Zaten firmada kimlik okuyucuları bulunduğu için bunun kolay bir değişiklik olacağını düşünüyor.

Bir kimlik okuyucusu ekliyoruz ve ücret bordrosuyla bağlantısını kuruyoruz.

Bu durumdayken, front panelin cash boxa kullanıcının ne kadar parasının olduğunu sorması doğru olmaz. Çünkü gizli bir bilgidir. Bunun yerine kahvenin ücretini verir ve "Sende 25 cent var mı" diye sorar.

Çeşitler

İnsanlar kahve yerine latte alıyor. Makinenin yeni içecekler eklenmeye ve fiyatlarının her an değiştirilmesine müsait olmasını istiyoruz. Espresso, cuppuccino... ekleyeceğiz.

Büyük Front Panel nesnesini parçalamalıyız.

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\18.gif

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\19.gif

Sistemin kalitesi

Bir tasarımı iyi yapan nedir?

Bir kere, her bir tasarım kendi zamanında iyiydi. Her biri nesnelerle çalışıyordu.

Aralarındaki fark, destekledikleri geleceklerdi.

İlk tasarımda fiyatların değişeceğini öngörmemiştik.
İkinci tasarımda kredi kartlarını öngörmemiştik.
Dördünce tasarımdaysa sistemin geneli, fiyatlara ve reçetelere bağlı deildi.

Bu durumda, bir tasarımın kalitesini tartışamayız ta ki belirli bir geleceğin muhtemel olduğuna karar vermeden.

Kaliteli tasarıma karar veren 6 tane kriter (veya test) var:

1. Veri bağlılık testi: Nesnelerden beklenen işleri gerçekleştirmeleri için gerekli tüm bilgiyi toplayabiliyor musun?

2. Soyutluk testi: Nesnenin ismi soyut anlamını taşıyor mu? Soyutlama doğal bir anlama sahip mi? Uzmanların dilinde kullanılan bir şey mi?

3. Sorumluluk uyumu testi: İsim, ana sorumluluk ifadesi, veri ve fonksiyonlar birbiriyle uyumlu mu? Çoğu zaman veri ve fonksiyonlar ilk ikisini aşar. Bu durumda nesneyi bölmek veya soyutlamayı tekrar düşünmek gerekebilir.

4. Veri çeşitlenme testi: Her türlü veri tipini desteklemek.

5. Evrim testi: İş kurallarında, teknolojide ve servislerdeki değişimleri tasarımın nasıl ele alacağı. Ne kadar unsur değişecek?

6. İletişim desenleri testi: Kötü iletişim desenlerinin oluşup oluşmadığını kontrol eder. Özellikle, çemberlere bakar.

Mesela ilk üç tasarım soyutluk ve sorumluluk testlerini geçemez. Herhangi bir soyutlama yok, direk makine parçaları kullanılmış.

Sorumlulukları birleştirmek

Dördüncü tasarım nereden geldi?

Şunu sorduk: 1.25 dolarlık bir moccha eklemek için kaç komponenti değiştirmeliyiz? Cevap, iki. (Niye? Sadece Front Panel) Sonra tek bir komponentin etkilenmesi için ne yapmalıyız, diye sorduk.

Bu soruyu yanıtlamak için, hangi sorumlulukların etkilendiğini bulduk. Fiyat ve Reçete. Bu ikisi farklı nesnelerdeydi.

Kahve Makinesi Uygulama

Mikser - Ürün hazırlama

Mikser:
1. Mikser -> Kutular (bosalt)

Baslangic:

OnPanel -> Mikser.hazirla (urun)

mikser -> urun.hazirla(kendi) --- Strateji

urun -> mikser.bosalt("Seker") --- Komut

mikser -> sekerKutusu.bosalt

Hazirlanan nesne bir kahve olacak. Bunu nasil test edecegiz? Dogru nesne hazirlandi mi acaba?

Mesela sekerli, kremali kahve istediysek, sundan emin olmaliyiz. Seker kutusuna, kahve kutusuna, krema kutusuna ve su haznesine bosalt emri gitmis olmali.

Buradaki testin zorlugu surada, uretilen nesne aslinda harici bir nesne, bilgi sisteminin disinda. Bunu olcme imkanimiz dogrudan yok.

Secenekler:
1. Kutularin icindeki miktarlari kontrol edebiliriz. Ama bunun icin ek kodlama yapmamiz gerekecek. Olcum aleti misali bir seyi kodun icine yerlestirmemiz gerekecek.

2. Kutulara bosalt emri gidince bir olay uretirler, biz de onlara abone oluruz. Bu makul gorunuyor, cunku gercek dunyada da boyle bir sey olmali. Hata kontrolu ve kullanici yonlendirme acisindan. file:///?Kitaplar|test%20kahve%20hazırlama|196|0

Seçenek pasifleştirme

Seçenekler aktif veya pasif durumda olacak. Öyleyse, seçenekleri artık birer string olarak yürütemeyiz.

String komut -x Enum komut

String komutları tip güvenli enumlara çevir. file:///?Kitaplar|Enum%20yapısı|2782|0

```
    public void bosalt(String malzeme) {
        if( malzeme == "şeker")
            sekerKutusu.bosalt(this);
```

Sonra:

```
    public void bosalt(Malzeme malzeme) {
        if( malzeme == Malzeme.SEKER)
            sekerKutusu.bosalt(this);
    }
```

Enum:

```
public class Malzeme {
    private final String isim;

    private Malzeme(String isim) { this.isim = isim; }

    public String toString()  { return isim; }

    public static final Malzeme SEKER           = new Malzeme("şeker");
    public static final Malzeme KREMA           = new Malzeme("krema");
    public static final Malzeme SU              = new Malzeme("su");
    public static final Malzeme TURK_KAHVESI    = new Malzeme("türk kahvesi");
    public static final Malzeme CAPPUCCINO      = new Malzeme("instance");
    public static final Malzeme LATTE           = new Malzeme("latte");
}
```

Ürün seçme

Fabrika metodu ve komut kalıbı birlikte.

Önce:

```
secim = kahveMakinesi.sec("Cappuccino");
```

Sonra:

```
secim = kahveMakinesi.sec(Cappuccino.instance());
```

instance metodu:

```
    protected static Urun instance;

    public static Urun instance(){
        if( instance == null )
            instance = new Cappuccino();
        return instance;
    };
```

sec metodu:

Önce:

```
    public Urun sec(String secim) throws YetersizPara {
        if( gecersizSecim(secim) )
            throw new GecersizSecim();
        Urun secilenUrun ;

        if( secim == "Cappuccino")
            secilenUrun = new Cappuccino();
        else if( secim == "Latte")
            secilenUrun = new Latte();
        else
            secilenUrun = new TurkKahvesi();

        paraKutusu.cek(secilenUrun);
        mixer.hazirla(secilenUrun);
        return secilenUrun;
    }
```

Sonra:

```
    public Urun sec(Urun secilenUrun) throws YetersizPara {
        paraKutusu.cek(secilenUrun);
        mixer.hazirla(secilenUrun);
        return secilenUrun;
    }
```

test kahve hazırlama

2. Kutulara bosalt emri gidince bir olay uretirler, biz de onlara abone oluruz. Bu makul

gorunuyor, cunku gercek dunyada da boyle bir sey olmali. Hata kontrolu ve kullanici yonlendirme acisindan.

Gözlemci kalıbı. file:///?Kod%20Parcalari|PropertyChange|1025|0

kutu -> bosalt { olay firlat }

sekerGozlemci { olayi yakala }

Fırlatma Kodu:

```
    public void bosalt(Mixer mixer) {
        if( icerik == 0 )
            throw new BosKutuException();
        changeSupport.firePropertyChange( getMalzeme().toString(),icerik,--icerik);
        if( icerik == 0 )
            mixer.bosaldi(this);
    }
```

Yakalama Kodu:

```
    public void testKahveHazirlama() throws YetersizPara {
        final int[] sekerUyari = new int[] { 0 };
        PropertyChangeListener sekerGozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                sekerUyari[0]++ ;
            }
        };
        Kutu sekerKutusu = makine.getSekerKutusu();
        sekerKutusu.addPropertyChangeListener(sekerGozlemci);
        makine.paraAt(75);
        makine.sec("Cappuccino");
        assertEquals(1, sekerUyari[0]);
    }
```

İki kutu için:

```
    public void testKahveHazirlama() throws YetersizPara {
        final int[] sekerUyari = new int[] { 0 };
        PropertyChangeListener sekerGozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                sekerUyari[0]++ ;
            }
        };
        final int[] capUyari = new int[] { 0 };
        PropertyChangeListener capGozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                capUyari[0]++ ;
            }
        };
        Kutu sekerKutusu = makine.getSekerKutusu();
        Kutu capKutusu = makine.getCappuccinoKutusu();
        sekerKutusu.addPropertyChangeListener(sekerGozlemci);
        capKutusu.addPropertyChangeListener(capGozlemci);
        makine.paraAt(75);
        makine.sec("Cappuccino");
        assertEquals(1, sekerUyari[0]);
        assertEquals(1, capUyari[0]);
    }
```

Sadeleştirme:

Olaylara farklı isimler veririz. (getPropertyName)

```
    public void testKahveHazirlama() throws YetersizPara {
        final int[] sekerUyari = new int[] { 0 };
        final int[] capUyari = new int[] { 0 };
        PropertyChangeListener gozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                if ( evt.getPropertyName().equals(Malzeme.SEKER.toString()) ) {
                    sekerUyari[0]++ ;
                }
                if ( evt.getPropertyName().equals(Malzeme.CAPPUCCINO.toString()) ) {
                    capUyari[0]++ ;
```

```
                }
            }
        };
        Kutu sekerKutusu = makine.getSekerKutusu();
        Kutu capKutusu = makine.getCappuccinoKutusu();
        sekerKutusu.addPropertyChangeListener(gozlemci);
        capKutusu.addPropertyChangeListener(gozlemci);
        makine.paraAt(75);
        makine.sec(Cappuccino.instance());
        assertEquals(1, sekerUyari[0]);
        assertEquals(1, capUyari[0]);
    }
}
```

AbstractKutu

Kutu:

```
public interface Kutu {
    public void bosalt(Mixer mixer);
```

Şeker Kutusu:

```
public class SekerKutusu implements Kutu {
    protected int icerik = 30;

    protected PropertyChangeSupport changeSupport;

    public synchronized void removePropertyChangeListener( PropertyChangeListener changeListener) {
        changeSupport.removePropertyChangeListener(changeListener);
    }
    public synchronized void addPropertyChangeListener( PropertyChangeListener changeListener) {
        changeSupport.addPropertyChangeListener(changeListener);
    }

    public SekerKutusu() {
        changeSupport = new PropertyChangeSupport(this);
    }

    public void bosalt(Mixer mixer) {
        if( icerik == 0 )
            throw new BosKutuException();
        changeSupport.firePropertyChange( "şeker",icerik,--icerik);
        if( icerik == 0 )
            mixer.bosaldi(this);
    }
}
```

Benzer şekilde Cappuccino kutusu vs. tanımlanır.

Bunlarda değişen bir tek yer var. O da fırlatılan olayın ismi.

Dolayısıyla orası hariç her yeri bir üst sınıfa taşıyabiliriz.

```
public abstract class AbstractKutu implements Kutu {
    protected int icerik = 30;

    protected int isim ;

    protected PropertyChangeSupport changeSupport;

    public synchronized void removePropertyChangeListener( PropertyChangeListener changeListener) {
        changeSupport.removePropertyChangeListener(changeListener);
    }
    public synchronized void addPropertyChangeListener( PropertyChangeListener changeListener) {
        changeSupport.addPropertyChangeListener(changeListener);
    }

    public AbstractKutu() {
        changeSupport = new PropertyChangeSupport(this);
    }

    public void bosalt(Mixer mixer) {
```

```
        if( icerik == 0 )
            throw new BosKutuException();
        changeSupport.firePropertyChange( isim ,icerik,--icerik);
        if( icerik == 0 )
            mixer.bosaldi(this);
    }
}

public class SekerKutusu implements Kutu {
    public SekerKutusu() {
        isim = "şeker";
    }
}
```

Ancak bu kadar basit bir sorumluluk için bir sınıf tanımlamaya da gerek yok.

```
public abstract class AbstractKutu implements Kutu {
    protected int icerik = 30;

    protected abstract Malzeme getMalzeme();

    protected PropertyChangeSupport changeSupport;

    public synchronized void removePropertyChangeListener( PropertyChangeListener changeListener) {
        changeSupport.removePropertyChangeListener(changeListener);
    }
    public synchronized void addPropertyChangeListener( PropertyChangeListener changeListener) {
        changeSupport.addPropertyChangeListener(changeListener);
    }

    public AbstractKutu() {
        changeSupport = new PropertyChangeSupport(this);
    }

    public void bosalt(Mixer mixer) {
        if( icerik == 0 )
            throw new BosKutuException();
        changeSupport.firePropertyChange( getMalzeme().toString(),icerik,--icerik);
        if( icerik == 0 )
            mixer.bosaldi(this);
    }
}
```

AbstractKutu tipinden her bir nesne oluşturulurken, getMalzeme metodu da tanımlanır:

```
    private Kutu sekerKutusu = new AbstractKutu(){
        protected Malzeme getMalzeme() {
            return Malzeme.SEKER;
        }
    };
```

Exceptionlar

Checked exceptionlar, kullanıcı girişindeki istisnaları veya hataları kontrol içindir.

Runtime exceptionlar, programlama hatalarını önlemek içindir.

Örnek,

Kullanıcı 35 TLye kahve alabilir. Ancak 20 TL attıktan sonra kahve düğmesine basıyor. Bu ne tip bir hatadır?

Kullanım hatasıdır. Daha doğrusu sistemin normal kullanım şekline bir istisnadır.

Peki OnPanel sınıfının sec(String secim) diye bir metodu var. Programcımız, sec metoduna Coppuccino diye bir secim gönderdi. Bu ne tip bir hatadır?

Programlama hatasıdır. sec metodunun varsayımlarından biri girdilerin {Cappuccino, Türk Kahvesi, Latte} kümesinin bir üyesi olması şeklindedir. Yanlış girilen bir seçenek, sec metodunun varsayımını ihlal eder. Bu yüzden de hata üretir. Ancak bu hata kullanıcıyı ilgilendirmez, dolayısıyla ona yansıtılmaz.

Programlama hataları, RuntimeExceptiondan türer.
Kullanım hataları, Exceptiondan türer.

Şu hataların tiplerini söyleyin:

```
    public Urun sec(Urun secilenUrun) throws YetersizPara {
        paraKutusu.cek(secilenUrun);
        mixer.hazirla(secilenUrun);
        return secilenUrun;
    }
```

Kullanım hataları mutlaka ya yakalanmalı ya da fırlatılmalıdır.

```
    public void bosalt(Mixer mixer) {
        if( icerik == 0 )
            throw new BosKutuException();
        changeSupport.firePropertyChange( getMalzeme().toString(),icerik,--icerik);
        if( icerik == 0 )
            mixer.bosaldi(this);
    }

    public boolean cek(Urun urun) throws YetersizPara{
        if( para >= urun.fiyat() ){
            setPara(para - urun.fiyat()) ;
            return true;
        }
        else
            throw new YetersizPara();
    }
```

Yetersiz para atan kullanıcının uyarılması gerekir:

```
        cappuccinoSecim = new AbstractAction("Cappuccino"){
                    public void actionPerformed(ActionEvent e) {
                        Urun secim;

                        try {
                            secim = kahveMakinesi.sec(Cappuccino.instance());
                            resim.setIcon(new ImageIcon(capURL));
                        } catch (YetersizPara yetersizPara) {
                            JOptionPane.showMessageDialog(frame,
                                    "Para yetersiz.\n   Fiyat: " + (new Cappuccino()).fiyat() +
                                    "\n   Atilan Para: " + kahveMakinesi.atilanPara());
                        }
                    }
                };
```

statik metodu üst sınıfa taşı

Fabrika metodu instanceı tek tek her bir üründe tanımlamak yerine AbstractUrunde tanımlamak istiyoruz.

Cappuccino:

```
    public static Urun instance(){
        if( instance == null )
            instance = new Cappuccino();
        return instance;
    };
```

Ancak her bir Ürün sınıfında instance metodunu bu şekilde tanımlamamız gerekiyor.

Bunun yerine:

AbstractUrunde tanımlayalım.

```
    public abstract static Urun instance();
```

Ancak statik bir metot soyut olamaz.

```
    public static Urun instance(){
        if( instance == null )
            instance = this.getClass().newInstance();
        return instance;
```

```
    };
```

Ancak this´e statik bir bağlamdan erişmek mümkün değildir.

Dolayısıyla bu metodu her bir sınıf için tanımlamak gerekiyor.

toString

```
public abstract class AbstractUrun implements Urun {
    protected String kahveTipi;

    public String toString() {
        return kahveTipi;
    }

---
public class TurkKahvesi extends AbstractUrun{
    public TurkKahvesi() {
        kahveTipi = "Türk Kahvesi";
    }
```

kutuları boşaltma

```
public class Mixer {
    public void bosalt(Malzeme malzeme) {
        if( malzeme == Malzeme.SEKER)
            sekerKutusu.bosalt(this);
        if( malzeme == Malzeme.CAPPUCCINO)
            cappuccinoKutusu.bosalt(this);
    }

---
```

Yukarıdaki if koşullamaları nasıl basitleştirilebilir?

- HashMap
- Polimorfizm, ziyaretçi, double dispatch?

BosKutuException

```
public abstract class AbstractKutu implements Kutu {
    public void bosalt(Mixer mixer) {
        if( icerik == 0 ){
            throw new BosKutuException();
        }
        changeSupport.firePropertyChange( getMalzeme().toString(),icerik,--icerik);
        if( icerik == 0 )
            mixer.bosaldi(this);
    }
```

Boş bir kutuya boşalt komutu gitmemeli. boşalt metodunun bu önkoşulu.

Eğer kutu boşken, bu komutu gönderirsek bu bir programlama hatası olur. Programlama hatalarının yakalanmaması gerekir. Bu hatalar programı çökertmeli ki, hemen düzeltilebilsin.

Boşalma

OnPanel -> Mixer -> Kutu . bosalt

Eger kutu tamamiyle bosalirsa, kutu bosaldigi uyarisinin obur yonde gitmesi gerekiyor.

Kutu -> Mixer -> OnPanel . kutuBosaldi

Ancak bunu yapabilmek icin ya iki yonlu referans bulunmali ya da bosalt mesaji OnPanel nesnesini parametre olarak gondermeli.

Bunlarin ikisi de arayuzu cok bozuyor. Bunun yerine, olay firlatalim.

Peki yeni bir olay nesnesi mi tanimlamaliyiz, yoksa mevcut olayi mi kullanmaliyiz?

changeSupport.firePropertyChange(getMalzeme().toString(),icerik,--icerik);

Kavramsal olarak farkli bir olay tipi, maksadi daha iyi ifade etmek icin yeni bir nesne tanimlamaliyiz.

Anlasilma, birkac satir fazla yazip okumaktan daha degerlidir. Hem istemci tarafindaki yuku azaltir.

Ancak kolay yoldan gidelim, ileride ihtiyac olursa bu degisikligi yapariz.

```
public abstract class AbstractUrun implements Urun {
    private Map recete = new HashMap();

    public void kutuBosaldi(String kutu) {
        if( recete.containsKey(kutu) )
            aktif = false;
    }
```

cast hatası

```
        mixer.getCappuccinoKutusu().addPropertyChangeListener(Cappuccino.instance());
```

Şöyle bir hata veriyor:

```
    Error:  line (22) addPropertyChangeListener(java.beans.PropertyChangeListener) in
versiyon2.kutu.Kutu cannot be applied to (versiyon2.urun.Urun)
```

Dikkat ederseniz, instance() metodu Urun tipinden bir nesne döndürüyor. Halbuki AbstractUrun sınıfı aslında PropertyChangeListener interfaceini gerçekleştiriyor.

```
public abstract class AbstractUrun implements Urun, PropertyChangeListener{
    public void propertyChange(PropertyChangeEvent evt) {
        if( evt.getNewValue().equals(Integer.valueOf("0")))
            aktif = false;
    }
```

Urun interfaceindeyse, propertyChange metodu yok.

Dolayısıyla cast etmeliyiz:

```
        mixer.getCappuccinoKutusu().addPropertyChangeListener((PropertyChangeListener)Cappuccino.instan
ce());
```

Ürün – Kutu bağlantısı

Ürünler, Kutuları dinler. Eğer kutu boşalırsa bir olay fırlatır ve ürün bu olayı dinler ve kendisini pasif hale getirir. Böylece bu seçenek ön panelden seçilemez.

Bağlama:

```
        mixer.getCappuccinoKutusu().addPropertyChangeListener((PropertyChangeListener)Cappuccino.instan
ce());
```

Dinleme:

```
    public void propertyChange(PropertyChangeEvent evt) {
        if( ! ( evt.getSource() instanceof Kutu ) ) return ;
        if( ((Integer) evt.getNewValue()).intValue() == 0 )
            aktif = false;
    }
```

Kutu.bosalt() olayı

```
public abstract class AbstractKutu implements Kutu {
    public void bosalt(Mixer mixer) {
        if( icerik == 0 ){
            throw new BosKutuException();
```

```
        }
        changeSupport.firePropertyChange( getMalzeme().toString(),icerik,--icerik);
    }
```

Kutu nesnesine bosalt() mesajı gidince, olay fırlatılması lazım. Bunu test edelim.

```
    public void testKutuBosaltmaOlayi() throws YetersizPara {
        final boolean[] olayGerceklesti = new boolean[1];
        olayGerceklesti[0] = false;
        PropertyChangeListener gozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                olayGerceklesti[0] = true;
            }
        };
        Kutu capKutusu = makine.getCappuccinoKutusu();
        capKutusu.addPropertyChangeListener(gozlemci);
        Urun cap = Cappuccino.instance();
        makine.paraAt(cap.fiyat());
        makine.sec(cap);
        assertTrue( "Kutu.bosalt() olay firlatmadi!",olayGerceklesti[0] );
    }
```

Pasifleştirme

Pasifleştirme nasıl olacak?

Ön Panele pasifleştir komutu gelince, artık ilgili ürünün seçilmesine izin vermeyecek.

Bu ne demek oluyor?

1. Kullanıcı arayüzünde o butonu pasif yapacak.
    Kullanım hatası. Exception.
    Programın içinde komut verilirse, o zaman ürün hazırlanır.

2. seç metodu o seçeneğin seçilmesi durumunda hata üretecek.
    Programlama hatası. Runtime Exception.

Eğer programlama hatası olursa, seç metodu çağrılmadan önce aktif seçeneklerin öğrenilmesi gerekir. Bu da işi biraz zahmetli hale getirir.

3. Kullanıcı arayüzünde buton pasif yapılır. Eğer programın içinden bu komut verilirse ürünün hazırlanmaması lazım. Bu yüzden de özel bir durum oluşturulur. Bu durumdayken, o ürün hazırlanmaz.

Demek doğru çözüm, durum kalıbı olacak. Niye?

Çünkü sistem, belli bir olaya bazen belli bir tepkiyi veriyor, bazen öbür tepkiyi veriyor.

Ancak her iki tepki de normal tepkiler. İstisnai olaylar değil.

Seçeneği pasifleştirme

Bir kutu boşaldığında bu kutuyu kullanan ürünlerin seçilmesini pasif hale getirmeliyiz. Yani önPanel.sec() metodu bir BosUrun döndürmeli.

```
    public void testPasifSecenek() throws YetersizPara {
        Urun cap = Cappuccino.instance();
        // kutuyu bosalt
        final int[] kalanMiktar = new int[] { 0 };
        PropertyChangeListener gozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                kalanMiktar[0] = ((Integer)evt.getNewValue()).intValue();
            }
        };
        Kutu capKutusu = makine.getCappuccinoKutusu();
        capKutusu.addPropertyChangeListener(gozlemci);
        while( kalanMiktar[0] >= 0 ){
            makine.paraAt(cap.fiyat());
            makine.sec(cap);
```

```
            System.out.println("kalanMiktar = " + kalanMiktar[0]);
            if(kalanMiktar[0] == 0) break;
        }
        makine.paraAt(cap.fiyat());
        assertEquals( BosUrun.instance() , makine.sec(cap) );
    }
```

Testler arası etkileşim

Unit testingin temel kuralı, testlerin birbirini etkilememesidir. Öbür türlü testlerin güvenilirliği azalır.

Ancak bizim testlerimizde bu olmuyor. Pasifleştirme testiyle, kutu boşaltma olayı testi birbiriyle etkileşim halinde. Eğer bu iki testi birlikte çalıştırırsak, kutu boşaltma testi hata veriyor. Ancak pasifleştirme testini çalıştırmazsak, kutu boşaltma testi doğru sonuç veriyor.

```
    public void testPasifSecenek() throws YetersizPara {
    public void testKutuBosaltmaOlayi() throws YetersizPara {
```

Niçin?

testPasifSecenek içinde cappuccino kutusunu boşaltıyoruz. testKutuBosaltmaOlayi içinde yeni bir cappuccino istediğimizde doğal olarak bunu üretmiyor.

setUp metodunda yeni bir KahveMakinesi nesnesi oluşturuyoruz. Her test metodunun öncesinde bu metot yeniden çağrılır. Ayrıca her test metodunun sonunda da tearDown() metodu çağrılır.

Bizim durumumuzda her seferinde yeni bir KahveMakinesi nesnesi oluşturmamıza rağmen, niye testler birbirine bağımlı?

Çünkü testPasifSecenek içinde Cappuccino ürününü pasif hale getiriyoruz. Ürün nesneleri birer Singleton olduğu için, bu KahveMakinesi her üretildiğinde yeni bir Cappuccino üretilmiyor. Dolayısıyla bundan sonraki testlerin tümünde pasif kalıyor.

Düzeltmek için, Ürün sınıflarına instance()ın tersi olan statik bir release metodu koy. Bu instance alanını null yapsın.

```
    public static void release(){
        instance = null;
    }

    protected void tearDown() throws Exception {
        Cappuccino.release();
    }
```

Ürünlerle Kutuları Bağlama

Bir ürün birçok kutuya ihtiyaç duyuyor. Bir kutudaki malzeme bitince, buna ihtiyaç duyan ürünlerin pasifleştirilmesi lazım. Dolayısıyla ürünlerin, ihtiyaç duydukları kutuları dinlemesini sağlamalıyız.

Bunu nasıl yapacağız. Bir çözüm:

```
        Map kutularMap;
        kutularMap = mixer.getKutularMap();
        for (int i = 0; i < urunler.size(); i++) {
            Urun urun = (Urun) urunler.get(i);
            for (Iterator iter=kutularMap.entrySet().iterator(); iter.hasNext(); ) {
                Map.Entry e = (Map.Entry) iter.next();
                Kutu kutu = (Kutu) e.getValue();
                Malzeme malzeme = (Malzeme) e.getKey();
                if( urun.contains( malzeme ){
                    kutu.addPropertyChangeListener( ((PropertyChangeListener) urun));
                }
            }
        }
```

```
        List kutular;
        kutular = mixer.getKutular();
        for (int i = 0; i < urunler.size(); i++) {
```

```
            Urun urun = (Urun) urunler.get(i);
            for (int j = 0; j < kutular.size(); j++) {
                Kutu kutu = (Kutu) kutular.get(j);
                if( urun.gereksinir(kutu) ){
                    kutu.addPropertyChangeListener(((PropertyChangeListener) urun));
                }
            }
        }
```

Hangisi daha iyi? Bir fark var mı?

List ile yapmak daha kolay. Ama ileride gereksiniriz diye Map yapma ihtiyacı hissediyorum.

Reçeteyi deklaratif tanımla

Reçete, farklı ürünler arasında değişebiliyor. Bunun programatik olarak değil, deklaratif olarak tanımlanabilmesini sağlamalıyız.

Önce reçeteyi bir liste olarak yürütelim:

```
    public void testDeklaratifRecete() throws YetersizPara {
        final boolean[] olayGerceklesti = new boolean[1];
        olayGerceklesti[0] = false;
        PropertyChangeListener gozlemci = new PropertyChangeListener(){
            public void propertyChange(PropertyChangeEvent evt) {
                olayGerceklesti[0] = true;
            }
        };

        List recete = new ArrayList();
        recete.add(Malzeme.CAPPUCCINO);
        recete.add(Malzeme.SEKER);
        recete.add(Malzeme.SU);
        cappuccino.setRecete(recete);

        cappuccino.receteDeklaratif(mixer);         recete metodu yerine

        Ozne capKutusu = makine.getCappuccinoKutusu();
        capKutusu.addPropertyChangeListener(gozlemci);

        makine.paraAt(cappuccino.fiyat());
        makine.sec(cappuccino);

        assertTrue("Kutu bosaltma gerceklesmedi", olayGerceklesti[0]);
    }
```

AbstractUrunde tanımlanır:

```
    public void receteDeklaratif(Mixer mixer) {
        for (int i = 0; i < recete.size(); i++) {
            Malzeme malzeme = (Malzeme) recete.get(i);
            mixer.bosalt(malzeme);
        }
    }
```

Reçeteyi xml dosyasından almak

XML işleme için: file:///?Kod%20Parcalari|JDom|13|0

XML dosyası:

```
<Tarif>
    <Urun isim="cappuccino">
        <Malzeme>"cappuccino"</Malzeme>
        <Malzeme>"su"</Malzeme>
        <Malzeme>"seker"</Malzeme>
    </Urun>
</Tarif>
```

Okuma denemesi testi:

```
    public void testReceteXml() throws JDOMException {
        SAXBuilder builder = new SAXBuilder();
        Document document = builder.build(new File("res/Recete.xml"));
        Element tarif = document.getRootElement();
        Element urun = tarif.getChild("Urun");
        String cappuccino = urun.getAttributeValue("isim");
        assertEquals("Tarif/Urun/@isim okunmadi","cappuccino",cappuccino);
    }
```

Sadece okuyup okumadığını test ediyoruz.

Gerçek okuma testi:

Reçeteyi properties dosyasından almak

Properties dosyası:

```
        cappuccino.malzemeler=cappuccino,su,şeker
```

Deneme testi:

```
    public void testReceteProperties() throws IOException {
        Properties parametreler = new Properties();
        FileInputStream in = new FileInputStream("res/Recete.properties");
        parametreler.load(in);
        in.close();

        String cappuccinoMalzemeler = parametreler.getProperty("cappuccino.malzemeler");
        Pattern pattern = Pattern.compile("\\,");
        String[] malzemeler ;
        malzemeler = pattern.split(cappuccinoMalzemeler);
        boolean test = Arrays.asList(malzemeler).contains("cappuccino");
        assertTrue("cappuccino okunmadi",test);
    }
```

Gerçek test:

```
    public void testReceteOkuma(){
        List recete = cappuccino.readRecete("res/Recete.properties");
        assertTrue("Recete listesi olusmadi",recete.contains(Malzeme.CAPPUCCINO));
    }
```

Ancak readRecete aslında private bir metot. Bunu nasıl test edebiliriz?

```
    public List readRecete(String file) {
        // todo: hata yakalama mekanizmasi
        try {
            Properties parametreler = new Properties();
            FileInputStream in = new FileInputStream("res/Recete.properties");
            parametreler.load(in);

            String cappuccinoMalzemeler = parametreler.getProperty("cappuccino.malzemeler");
            Pattern pattern = Pattern.compile("\\,");
            String[] malzemeler ;
            malzemeler = pattern.split(cappuccinoMalzemeler);
            for (int i = 0; i < malzemeler.length; i++) {
                String m = malzemeler[i];
                Malzeme malzeme = Malzeme.get(m);
                recete.add(malzeme);
            }
            return recete;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
```

Malzeme.get:

```
    public static Malzeme get(String malzeme) {
        if( malzeme == SEKER.toString() ) return SEKER;
        if( malzeme == KREMA.toString() ) return KREMA;
        if( malzeme == SU.toString() ) return SU;
        if( malzeme == TURK_KAHVESI.toString() ) return TURK_KAHVESI;
        if( malzeme == CAPPUCCINO.toString() ) return CAPPUCCINO;
        if( malzeme == LATTE.toString() ) return LATTE;
        return null;
    }
```

Bu aslında bir map gibi çalışıyor. Bunun yerine statik bir map alanı tanımlamak istedim. Ancak initialization problemi çıktı. Statik initializer, enum nesneleri oluşturulmadan önce çalıştırıldığı için hata oluşuyor.

Koku:

Bir bilginin sadece tek bir temsili olmalı. Ancak mesela "cappuccino" hem properties dosyasında var, hem de CAPPUCCINO nesnesinde. Ayrıca eşleştirmeler de bir başka sorun.

Sadeleştirmeler

Kalıplar

Reçete -> command
Boşalma -> observer
Ürün seçme -> komut