BlackMamba: A Swing Case Study

BlackMamba: A Swing Case Study

file:///F:\Belgelerim\Yayınlar\Java\Makaleler\Kullanıcı%20Arayüzü\Yöntem\BlackMamba%20A%20Swing%20Case%20Study\BlackMamba%20A%20Swing%20Case%20Study.doc

BlackMamba <http://www.JavaForU.com>

http://www.onjava.com/lpt/a/4642

Burada Swing ile nesne odaklı tasarımın ilkelerini kullanarak bir örnek uygulamayı göreceğiz.

Uygulama basit bir HelloWorld uygulaması türünde değil. Ciddi. Internetten gelen mailleri preview etmek için kullanılacak. Böylece spam mailleri ayırt etmek mümkün olacak. Dolayısıyla uygulamanın bir ayağı networking gerektirecek. Ayrıca mail ayarlarıyla ilgili birçok dosyayı da işlemesi gerektirecek. Bu da başka bir komplekslik getirecek.

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\43.gif

Katmanlar

Katmanlar:

MVC yaklaşımına göre, model, sunum (view) ve kontrol (control) katmanları birbirinden bağımsızdır. En başta modeli inşa ederiz. Sonra sunum ve kontrolu buna bağlayabiliriz. Ancak gerçekte bu pek böyle değil. Önce sunumu tasarlarsak, modelin sunacağı fonksiyonaliteyi daha iyi anlayabiliriz. Yani en başta kullanıcı arayüzünün kağıt üstünde bir prototip tasarımını yapalım.

Hızlı uygulama geliştirmek için, bir UI Editör ile kullanıcı arayüzünü tasarlamalıyız. NetBeans´in ve Eclipse´in bedava arayüz tasarlayıcıları kullanılabilir. Bazı UI Editörler ürettikleri kodun içine metotlar ekliyorlar. Bu metotları kaldır. Sunum kodu, tamamıyla işlevsiz olsun. Yine oluşturulan örnek değişkenlerinin tümünü public olarak beyan et. Normalde bu nesne odaklılığın kapsülleme ilkesine aykırıdır. Ancak, kullanıcı arayüzünün komponentleri için bu geçerli olmamalı. Çünkü kapsüllemenin temelinde nesnelere sorumluluk yüklemek, ham veriyle iş yapmak yerine, nesnelere iş yaptırmak vardır. Halbuki, görsel komponentler, zaten sorumlulukları ham veri taşımaktır. Yani kapsüllenecek bir şey yok.

Bu Demeter Kanununa aykırı (Yani sadece kendi en yakın dostlarınla konuş ilkesine). Kullanıcı arayüzü dışında bu kurala bağlıyız.

Bütün sunum kodunu, ayrı bir pakette tut. Mesela, blackmamba.ui gibi. Aynı şekilde kontrol sınıflarını ayrı bir pakette tut: blackmamba.ui.control gibi. Kontrol sınıfları, kullanıcı arayüzünün mantığını tutarlar.

Dikkat: Kullanıcı arayüzü mantığı (bazen uygulama mantığı, yani application logic de denir) iş mantığından ayrı. Arayüz mantığı, sadece uygulamayla kullanıcının etkileşimine ait şeylerdir. Mesela, Kalem Ekle düğmesine tıkladığında, yeni bir pencere açılır. Bu açılan pencerede, Yardım düğmesine basınca, ilgili yardım gelir.

Kontrol sınıflarına fiil adları vermek daha yerinde olur (Başla, Giriş, KonfigüreEt gibi). Sunum sınıflarına ise isimler vermek uygun.

Kontrol sınıflarına fiil adları vermek, bir hikayeyle ilgili fonksiyonalitenin nereye konulacağını daha kolay tespit etmemizi sağlar.

Sunum, kontrol sınıfları tarafından kontrol edilir ve yönetilir. Sunum sınıfları kendi içlerinde herhangi bir iş yapamaz. Tüm işlemler dışarıdan yapılır. Sunuma, kontrol dışından erişimi engellemek için, **Macker** adlı bir araç kullanılabilir. Bu araçla, bu tarz mimari kararları mecbur tutmak sağlanır.

Bunun için Macker´ın konfigürasyon dosyasında şöyle bir kod yazılır:

```
<?xml version="1.0"?>
<macker>
  <ruleset name="Layering rules">

    <pattern name="view"
     class="blackmamba.ui.view.**" />
    <pattern name="control"
```

```
     class="blackmamba.ui.control.**" />

    <access-rule>
        <message>Only the Control can talk
                 to the View
        </message>

        <deny>
            <to pattern="view" />
            <allow>
                <from pattern="view"/>
            </allow>
            <allow>
                <from pattern="control"/>
            </allow>
        </deny>
    </access-rule>

  </ruleset>
</macker>
```

Kontrol

Kontrol sınıfları sunum üzerinde iş yapar. Şunları yaparlar:

1. Sunum nesnelerini oluşturmak
2. Dinleyicileri bağlamak
3. Modeli bağlamak
4. İşlemin sonunda, sunumları kaldırmak.

Kontrol sınıflarının aşırı şişmesi çok sık gerçekleşir. Bunu önlemek için, kontrol sınıflarına isimlerinin işaret ettiklerinin dışında iş yüklememekte fayda var. Buradaki sorumluluklar:

Controller/Verb	View/Noun	Description
Start	AboutDlg, AboutWin, MambaFrm	Launch main frame. Setup menu action listeners and delegate Login action.
Login	MambaFrm, LoginDlg	Launch login dialog. Setup button action listeners and delegate Connect and Accounts actions.
Connect	SplashPnl	Launch connect dialog. Connect to the server, update progress, and setup cancel button action listener
Fetch	MailsPnl, MambaFrm, SplashPnl	Launch fetch dialog. Connect to the server, fetch mails, update progress, submit mails to MailProcessor, and setup cancel button action listener.

Genel uygulama olarak, kontrol sınıflarını incelt. Şişmelerine izin verme. Daha çok fiil keşfet. Fonksiyonaliteyi bunlara dağıt. 500-600 satırın üzerine çıkmasın kontrol sınıfları.

Duyarlı Arayüz

Duyarlı Arayüz:

Uzun zaman işlemler varsa, bir Vazgeç seçeneği eklemekte fayda var. (Network bağlantıları gibi)

Ayrıca bu süre içinde, arayüzün kullanıcıyla etkileşimine izin vermek için, bu uzun operasyonları ayrı bir zincirde (thread) gerçekleştir.

Swing tek zincirli bir yapıdadır. (Swingin zincir yapısıyla ilgili bilgi için: file:///?Kitaplar|Rethinking%20Swing%20Threading|1221|0)

Harici bir zincirden arayüz üzerinde doğrudan işlem yapamayız. Bunun yerine SwingUtilities.invokeLater(runnable) metodunu kullanmalıyız. Böylece yaptığımız işlem, EventQueue´ye eklenir. Ve zamanı geldiğinde swing zinciri tarafından çalıştırılır. Şu an içinde bulunduğun zincirin swingin kendi zinciri olup olmadığını öğrenmek için: javax.swing.SwingUtilities.isEventDispatchThread() metodunu kullan.

volatile: file:///?Kod%20Parcalari|volatile|2|9

Maliyetli işlemleri asla, actionPerformed metodunun içinde yürütme. Yeni bir zincir oluştur. Aksi taktirde, arayüz, kullanıcının davranışlarına karşı tepki vermez.

Bu tarz çok zincirli işlemler sık sık yapılıyorsa, performans açısından bir zincir havuzu (Thread Pool) kullanmalı. Yeni zincir oluşturmak maliyetli bir işlemdir.

Kod:

```
public class Connect
{
    //Note the "volatile" keyword.
    private volatile int progressStep;
    ...
    final Runnable target = new Runnable()
            {
                public void run()
                {
                    connect();                              2
                }
            };
    ...
    ...
    dialog.setDefaultCloseOperation(
        javax.swing.WindowConstants \
                            .DO_NOTHING_ON_CLOSE);

    dialog.addWindowListener(
        new java.awt.event.WindowAdapter()
        {
            public void windowClosing(
                    java.awt.event.WindowEvent evt)
            {
                cancel();
            }
        });
    ...
    ...
    LiteThreadPool threadPool =
                        Setup.getLiteThreadPool();
    PoolThread thread1 = threadPool.getThread1();
    thread1.execute(target);                                1
    ...
    ...
    protected void connect()                                3
    {
        final Runnable runnable = new Runnable()
        {
            public void run()                               3.2
            {
                //Update UI
                splashPnl.connectingPrg.setValue(
                                    progressStep);
            }
        };

        try
        {
            mailHelper.connect(accountDetails);             3.1
            progressStep = 1;
            SwingUtilities.invokeLater(runnable);

            mailHelper.getMessageCount();
            progressStep = 2;
            SwingUtilities.invokeLater(runnable);

            mailHelper.disconnect();
            progressStep = STEP_END;
            SwingUtilities.invokeLater(runnable);
        }
        catch (Exception e)
```

```
        {
            ...
        }
    }

    protected void cancel()
    {
        boolean success = true;

        try
        {
            if (progressStep != STEP_END)
            {
                mailHelper.immediateDisconnect();
                success = false;
            }
        }
        catch (Exception e)
        {
        ...
        }

        dispose(success);
    }
   ...
   ...
}
```

Uygulama akışı:

1. Önce bir zincir havuzundan, 1. zincir alınır (1. zincir, mail işlemleri için, 2. zincir diğer kontroller için).

2. Alınan zincirden, target Runnable nesnesi çağrılır. (swing dışı zincirdeyiz)

3. Bu da connect metodunu çağırır.

3.1. Bağlantı kurulmaya çalışılır. (Bu zaman alan bir iştir.)

3.2. Bağlantı kurulduktan sonra, progress bar güncellenir (swing zincirine yine dönüyoruz.)

Zincir havuzu

setDaemon ile zincirin bir daemon zinciri olduğunu belirtiyoruz. Eğer bütün zincirler daemona aitse, JVM uygulamadan çıkar. Bu metot zincir başlatılmadan çağrılmalı.

Zincir havuzu:

```
public class LiteThreadPool
{
    private PoolThread thread1;
    private PoolThread thread2;

    public LiteThreadPool()
    {
         thread1 = new PoolThread();
         thread1.setDaemon(true);    // daemon zinciri mi kullanıcı zinciri mi?
         thread1.start();                                          1

         thread2 = new PoolThread();
         thread2.setDaemon(true);
         thread2.start();
    }

    //Thread1 used by MailProcessor.
    public PoolThread getThread1()
    {
         return thread1;
    }

    //Thread2 used by control classes.
```

```
    public PoolThread getThread2()
    {
         return thread2;
    }
}

public class PoolThread extends Thread
{
    private Object lock = new Object();
    private Runnable target = null;

    public void run()                                           2
    {
    while (true)
    {

        synchronized (lock)
        {
            while (target == null)
            {
                try
                {
                    lock.wait();                                2.1
                }
                catch (InterruptedException e)
                {
                    if (Setup.DEBUG)
                    {
                        e.printStackTrace();
                    }
                }
            } //end inner while

            try
            {
                target.run();                                   2.2
            }
            //Note that Exception is being caught.
            //If not, RuntimeExceptions can get
            //through and kill the PoolThread.
            catch (Exception e)
            {
                if (Setup.DEBUG)
                {
                    e.printStackTrace();
                }
            }

            target = null;                                      2.3

        } //end sync

    } //end while
    }

    public void execute(Runnable r)                        3
    {
        synchronized (lock)
        {
            target = r;
            lock.notifyAll();
        }
    }
}
```

Burada Thread1 MailProcessor olarak, Thread2 de kontrol sınıfları olarak kullanılacak.

Burada gerçek anlamda bir havuz yok aslında. Eğer bir çağırıcı, hedefinin bir zincir içinde işletilmesini isterse ve o sırada zincir zaten işliyorsa, o zaman bunun bitmesini bekleyecek. Gerçek bir zincir havuzunda, Synchronized Queue veri yapısı içinde, birçok zincir nesnesi tutulur.

Mantık akışı:

```
1.         thread1.start();                                     1
```

Komutuyla zincir hayat döngüsüne başlar.

2. Hayat döngüsüne başlayan zincirin run() metodu çağrılır.

2.1. Henüz hiçbir runnable target konulmadığı için, zincir bekler.

```
lock.wait();                                    2.1
```

3. Dışarıdan: Dışarıdan birisi mesela bir Connect düğmesine bastığında bir runnable nesnesini execute() metoduna gönderir.

Zincir beklemede olduğu için, synchronized bloğuna giriş serbesttir.

Burada notifyAll yapılır. Ve beklemede olan zincir uyanır.

2.2. Uyanan zincir ilerler ve kendine gönderilen target runnable nesnesini çalıştırır.

Asenkron Işlemler

Asenkron İşlemler:

Zincirlerle birlikte ilginç ve can sıkıcı bir sorun ortaya çıkıyor: Bir zincir içinde yürütülen metottan, çağırıcı nesneye bir sonuç döndürmek hiç kolay değildir. Çünkü sonuç döndürülünceye kadar, çağırıcı kendisi işlemini bitirmiş dönmüş olabilir.

Bu yüzden çözüm, geri çağrı (callback) metotlarını kullanmaktır. Böylece asenkron yürütme sağlanabilir.

Ancak şimdi de başka bir sorun ortaya çıkıyor:

Mesela Worker1 sınıfı ThreadPool sınıfına bazı işler veriyor. Sonuçları döndürmesi için, ThreadPool, Worker1´in done(boolean) metodunu kullanıyor.

Bir süre sonra da Worker2 sınıfı ThreadPool sınıfına bazı işler veriyor. Sonuçları döndürmesi için, ThreadPool, Worker2´nin completed(boolean) metodunu kullanıyor.

Gelecekte de başka bir sınıf, Worker3 sınıfı ThreadPool sınıfına bazı işler veriyor. Sonuçları döndürmesi için, ThreadPool, Worker3´ün ayrı bir metodunu kullanıyor.

Dolayısıyla ThreadPool sınıfı sürekli değişime maruz kalıyor. Bu da hoş bir şey değil. Bu sorunu hafifletmek için, ThreadPool sınıfının kullanıcılarına Callback interfaceini implemente etme zorunluluğu getiriyoruz.

Burada uyguladığımız tasarıma, Bağımlılığın Tersine Çevrilmesi (Dependency Inversion DIP) ilkesi denir. Bağımlılıklar, soyutlamalar üzerine olmalı, somutlamalar üzerine değil.

```
public interface Callback
{
    public void call();
}
```

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\47.gif

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\48.gif

---

```
public class Fetch
{
    public void init(MambaFrm frm, MailsPnl pnl,
                                     boolean all)
    {
        ...

        final Runnable target = new Runnable()
        {
            public void run()
            {
                fetch();
            }
        };
```

```
        ...
        ...
        LiteThreadPool threadPool =
                       Setup.getLiteThreadPool();
        PoolThread thread1 =
                         threadPool.getThread1();
        thread1.execute(target);
        ...
    }

    ...
    protected void fetch()
    {
        ...
        MailProcessor mailProcessor =
                         Setup.getMailProcessor();
        ...
        mailProcessor.process(mailDetails);
        ...
    }
}

public class MailProcessor
{
    private MailList mails;
    private MailList possibleSpamMails;
    private MailList spamMails;
    ...
    ...

    public void classify(MailDetails mail,
                         boolean addToMailList)
    {
        ...
        switch (mail.getSenderStatus())
        {
            case MailDetails.SENDER_KNOWN :
                ...
                possibleSpamMails.addMail(mail);
                break;
            case MailDetails.SENDER_UNKNOWN :
                ...
                possibleSpamMails.addMail(mail);
                break;
            case MailDetails.SENDER_SPAMMER :
                ...
                spamMails.addMail(mail);
                break;
        }
        ...
    }
}

public class CustomTableModel extends
            AbstractTableModel implements MailList
{
    private Callback onMailAdd;
    ...
    public void addMail(MailDetails mailDetails)
    {
        ...
        fireTableRowsInserted(index, index);

        onMailAdd.call();
    }
}
```

Kod Okuma

Start

Her şey Start ile başlıyor. Bu bir kontrol sınıfı.

Genel bakış:

```
	public static void main(String args[]) throws Exception
	{
		Setup.init1();
		Setup.initLF();

		//Splash screen
		AboutWin aboutWin = new AboutWin();
		aboutWin.contentLbl.setIcon(Setup.getIcon());
		aboutWin.contentLbl.setText(ABOUT_STR);
		aboutWin.pack();
		aboutWin.setLocationRelativeTo(null);
		aboutWin.show();

		Setup.init2();

		new Start().init();

		aboutWin.requestFocusInWindow();
		aboutWin.setVisible(false);
		aboutWin.dispose();
	}
```

---

init1: Temel ayarlar

init1: Sıçrama ekranı, help dosyası, kök dizini gibi temel kurulumlar:

```
class Start ...
	public static void main(String args[]) throws Exception
	{
		Setup.init1();

class Setup ...
	public static void init1() throws Exception
	{
		//Root Dir
		String rootPath = new File(".").getAbsolutePath();
		rootPath = rootPath.substring(0, rootPath.length() - 1);
```

Burada kök dizin bulunuyor. Önce bir çöp (dump) dosya oluşturuluyor. Onun yolu alınıyor. Consolda bu dizinler:

```
rootPath = C:\java\blackmamba\BlackMamba_v1.1src\.
rootPath = C:\java\blackmamba\BlackMamba_v1.1src\
```

Devam:

```
		rootDir = new File(rootPath, DIR_NAME_RESOURCES);
```

File sınıfının iki farklı constructorı:

new File(".") Bu yeni bir dosya üretiyor, verilen adla.
new File(rootPath, DIR_NAME_RESOURCES) Burada ilk argüman, ikinci argümanın hangi dizinde bulunduğunu gösteriyor.

File sınıfı farklı işletim sistemlerindeki farklı dosya erişim mekanizmalarının üzerinde bir soyutlama sunar.

```
	public static final String DIR_NAME_RESOURCES = "resources";
```

Dolayısıyla new File(rootPath, DIR_NAME_RESOURCES) sonucunda, kök dizinde resources adlı bir dizin oluşturuldu. File nesneleri, dizin de olabilir, normal dosya da.

```
	public static final String FILE_NAME_IMAGEICON = "Snake.png";
```

Devam:

```
		//Image Icon
		String imageFile = new File(rootDir, FILE_NAME_IMAGEICON).getAbsolutePath();
		icon = new ImageIcon(imageFile);
```

Burada önce, Snake.png dosyasına ulaştık. Sonra da bunu bir imaj olarak ImageIcon nesnesine yükledik..

Dikkat ederseniz, new File(rootDir, FILE_NAME_IMAGEICON) çağrısında, ilk argüman (rootDir) bir File nesnesi. İkinci argüman, bir String.

Devam:

```
		//Help File
		String help = new File(rootDir, FILE_NAME_HELP).getAbsolutePath();
		helpFile = new File(help);
```

Burada:

```
	public static final String FILE_NAME_HELP = "Help.html";
```

init1 böylece en temel işleri yapar:

initLF

initLF:

```
class Setup ...
	public static void main(String args[]) throws Exception
	{
		Setup.init1();
		Setup.initLF();
```

Bu arada Setup, blackmamba paketi içinde.

```
	public static void initLF()
	{
		try
		{
	    String uiTheme = System.getProperty("ui");
	    if (uiTheme == null)
	    {
	       setDefaultUI();
	    }
	    else
	    {
	        UIManager.setLookAndFeel(uiTheme);
	    }
		}
		catch (Exception exception)
		{
			if (DEBUG)
			{
				exception.printStackTrace();
			}
```

```
                }
        }
```

Sıçrama ekranı

Sıçrama Ekranı:

main() Devam:

```
                //Splash screen
                AboutWin aboutWin = new AboutWin();
                aboutWin.contentLbl.setIcon(Setup.getIcon());
                aboutWin.contentLbl.setText(ABOUT_STR);
                aboutWin.pack();
                aboutWin.setLocationRelativeTo(null);
                aboutWin.show();

                Setup.init2();
```

Sıçrama ekranı (Splash screen):

```
package blackmamba.ui.view;
public class AboutWin extends javax.swing.JWindow
        public AboutWin()
        {
                super();
                initComponents();
        }

        private void initComponents()
        { //GEN-BEGIN:initComponents
                contentLbl = new javax.swing.JLabel();

                getContentPane().setLayout(new java.awt.BorderLayout(5, 5));

                addMouseListener(new MouseAdapter()
                {
                        public void mousePressed(MouseEvent e)
                        {
                                setVisible(false);
                                dispose();
                        }
                });

                getContentPane().add(contentLbl, java.awt.BorderLayout.CENTER);

                pack();
        } //GEN-END:initComponents
```

Dikkat ederseniz, resim yükleme:

```
                aboutWin.contentLbl.setIcon(Setup.getIcon());
```

Burada aboutWin.contentLbl ile doğrudan nesnenin alanlarına erişim yapılıyor. Public her şey. Çünkü kullanıcı arayüzündeki bütün komponentler, aslında zaten primitif veri üzerinde işlem yaptığı için, bunları kapsüllemenin bir anlamı yok.

Burada güzel bir yardımcı metot var:

```
                aboutWin.setLocationRelativeTo(null);
```

Bu metotla, sıçrama ekranının ekranın ortasına yerleştirilmesi sağlanıyor. Eğer argüman olarak başka bir komponent göndermiş olsaydık, o komponentin ortasına yerleşirdi.

init2: Konfigürasyon veritabanları

init2:

Konfigürasyon veritabanlarının yüklenmesi.

Burada güzel bir yaklaşım var. Temel yöntem üç adımda:

```
accountsDatabase = new AccountsDatabase();
accountsDatabase.setFileName(accountsDatabaseFile);
accountsDatabase.load();
```

catch: Yeni dosya oluşturma

Eğer dosya yüklenemezse, yeni dosya oluşturur:

```
try
{
        accountsDatabase.load();
}
catch (FileNotFoundException fnfe)
{
        new File(accountsDatabaseFile).createNewFile();
        accountsDatabase.setAccountDetails(new TreeSet(new AccountDetailsComparator()));
}
```

Akış

Genel bakış:

```
public static void main(String args[]) throws Exception
{
        Setup.init1();
        Setup.initLF();

        //Splash screen
        AboutWin aboutWin = new AboutWin();
        aboutWin.contentLbl.setIcon(Setup.getIcon());
        aboutWin.contentLbl.setText(ABOUT_STR);
        aboutWin.pack();
        aboutWin.setLocationRelativeTo(null);
        aboutWin.show();

        Setup.init2();

        new Start().init();

        aboutWin.requestFocusInWindow();
        aboutWin.setVisible(false);
        aboutWin.dispose();
}

public static void init2() throws Exception
{
        initDatabases();

        //LiteThreadPool
        liteThreadPool = new LiteThreadPool();

        //MailProcessor
        mailProcessor = new MailProcessor();

        //MailHelper
        mailHelper = new MailHelper();
}

protected static void initDatabases() throws IOException
{
```

```
		String accountsDatabaseFile = new File(rootDir,FILE_NAME_ACCOUNTS).getAbsolutePath();
		String addressesDatabaseFile = new File(rootDir,FILE_NAME_ADDRESSES).getAbsolutePath();
		String spammersDatabaseFile = new File(rootDir,FILE_NAME_SPAMMERS).getAbsolutePath();
		String subjectsDatabaseFile = new File(rootDir,FILE_NAME_SUBJECTS).getAbsolutePath();
		String sizesDatabaseFile = new File(rootDir, FILE_NAME_SIZES).getAbsolutePath();

		//AccountsDatabase
		accountsDatabase = new AccountsDatabase();
		accountsDatabase.setFileName(accountsDatabaseFile);
		try
		{
			accountsDatabase.load();
		}
		catch (FileNotFoundException fnfe)
		...
```

initDatabases metodunda, tek tek bütün konfigürasyon veritabanlarını yüklüyoruz. Bunlar düz metin olarak sistemde tutuluyor.

```
	public static final String FILE_NAME_ACCOUNTS = "AccountsDB.txt";
```

Yapılandırıcı boş:

```
	public AccountsDatabase()
	{
	}
```

load

Yükleme, load() içinde gerçekleşir:

```
	public void load() throws FileNotFoundException, IOException
	{
		super.load();
		Properties props = getProps();

		String numOfAliasesStr = props.getProperty(PROP_NUM_ALIASES, "0").trim();
		final int numOfAliases = Integer.parseInt(numOfAliasesStr);
		accountDetails = new TreeSet(new AccountDetailsComparator());

		//Starts from 1
		for (int i = 1; i <= numOfAliases; i++)
		{
			String alias = props.getProperty(PROP_ALIAS_NAME_PREFIX + i).trim();
			if (alias == null)
			{
				continue;
			}
			String serverAddr = props.getProperty(PROP_ALIAS_SERVER_ADDR_PREFIX + i).trim();
			int port =
				Integer.parseInt(props.getProperty(PROP_ALIAS_SERVER_PORT_PREFIX +
i).trim());
			String loginPwd = props.getProperty(PROP_ALIAS_LOGIN_PWD_PREFIX + i).trim();

			String[] arr = PasswordEncDecrypt.decode(loginPwd);

			AccountDetails aliasDetails = new AccountDetails();
			aliasDetails.setAlias(alias);
			aliasDetails.setServerAddress(serverAddr);
			aliasDetails.setServerPort(port);
			aliasDetails.setLogin(arr[0]);
			aliasDetails.setPassword(arr[1]);
			accountDetails.add(aliasDetails);
		}
	}
```

Ortak işler:

```
public abstract class PropertiesDatabase ...
	protected void load() throws FileNotFoundException, IOException
	{
```

```
		InputStream is = new FileInputStream(fileName);
		props = new Properties();
		props.load(is);
		is.close();
	}
```

Devam:

```
		String numOfAliasesStr = props.getProperty(PROP_NUM_ALIASES, "0").trim();
```

getProperty(PROP_NUM_ALIASES, "0") metodunun ikinci argümanı, varsayılan değer.

```
	private static final String PROP_NUM_ALIASES = "NumOfAliases";
```

Sondaki trim() metoduyla, alınan Stringin başındaki ve sonundaki beyaz spaceler kaldırılır.

Devam:

```
		accountDetails = new TreeSet(new AccountDetailsComparator());
```

Burada bir TreeSet oluşturuyoruz. Bu bir Set implementasyonudur. Bu Setin elemanlarının nasıl sıralanacağını gönderdiğimiz argümanla belirliyoruz.

```
public class AccountDetailsComparator implements Comparator
{
	public int compare(Object o1, Object o2)
	{
		AccountDetails accountDetails1 = (AccountDetails) o1;
		AccountDetails accountDetails2 = (AccountDetails) o2;

		return accountDetails1.getAlias().compareTo(accountDetails2.getAlias());
	}

}
```

Password okumak

Devam:

```
			String loginPwd = props.getProperty(PROP_ALIAS_LOGIN_PWD_PREFIX + i).trim();
			String[] arr = PasswordEncDecrypt.decode(loginPwd);
```

Ana ekran: MambaFrm

Start bir kontrol sınıfı. Başlangıç ekranını oluşturur:

```
	public static void main(String args[]) throws Exception
	{
		Setup.init1();
		Setup.initLF();

		//Splash screen
		AboutWin aboutWin = new AboutWin();
		aboutWin.contentLbl.setIcon(Setup.getIcon());
		aboutWin.contentLbl.setText(ABOUT_STR);
		aboutWin.pack();
		aboutWin.setLocationRelativeTo(null);
		aboutWin.show();

		Setup.init2();

		new Start().init();

		aboutWin.requestFocusInWindow();
		aboutWin.setVisible(false);
		aboutWin.dispose();
	}

	public void init()
```

```
        {
                mambaFrm = new MambaFrm();

                prepareMenus();

                mambaFrm.setExtendedState(JFrame.MAXIMIZED_BOTH);
                mambaFrm.mainPnl.add(new JLabel(Setup.getIcon()));
                mambaFrm.show();
        }
```

MambaFrm

MambaFrm bir sunum sınıfıdır. Tamamıyle içindeki komponentleri yerleştirmekle uğraşır:

```
public class MambaFrm extends javax.swing.JFrame
{

        /** Creates new form Mamba */
        public MambaFrm()
        {
                super();
                initComponents();
        }

        /** This method is called from within the constructor to
         * initialize the form.
         * WARNING: Do NOT modify this code. The content of this method is
         * always regenerated by the Form Editor.
         */
        private void initComponents()
        { //GEN-BEGIN:initComponents
                mainPnl = new javax.swing.JPanel();
                menuBar = new javax.swing.JMenuBar();
                actionMnu = new javax.swing.JMenu();
                loginMnuIt = new javax.swing.JMenuItem();
                logoutMnuIt = new javax.swing.JMenuItem();
                separator1 = new javax.swing.JSeparator();
                exitMnuIt = new javax.swing.JMenuItem();
                preferencesMnu = new javax.swing.JMenu();
                manageAcctsMnuIt = new javax.swing.JMenuItem();
                configureMnuIt = new javax.swing.JMenuItem();
                helpMnu = new javax.swing.JMenu();
                helpMnuIt = new javax.swing.JMenuItem();
                visitSiteMnuIt = new javax.swing.JMenuItem();
                aboutMambaMnuIt = new javax.swing.JMenuItem();

                setTitle("BlackMamba Mert");
                mainPnl.setLayout(new java.awt.GridLayout(1, 1, 5, 5));
                getContentPane().add(mainPnl, java.awt.BorderLayout.CENTER);

                actionMnu.setMnemonic('a');
                actionMnu.setText("Actions");
                loginMnuIt.setAccelerator(
                        javax.swing.KeyStroke.getKeyStroke(
                                java.awt.event.KeyEvent.VK_I,
                                java.awt.event.InputEvent.CTRL_MASK));
                loginMnuIt.setMnemonic('i');
                loginMnuIt.setText("Login");
                actionMnu.add(loginMnuIt);
                logoutMnuIt.setAccelerator(
                        javax.swing.KeyStroke.getKeyStroke(
                                java.awt.event.KeyEvent.VK_O,
                                java.awt.event.InputEvent.CTRL_MASK));
                logoutMnuIt.setMnemonic('o');
                logoutMnuIt.setText("Logout");
                actionMnu.add(logoutMnuIt);
                actionMnu.add(separator1);
                exitMnuIt.setAccelerator(
                        javax.swing.KeyStroke.getKeyStroke(
                                java.awt.event.KeyEvent.VK_Q,
                                java.awt.event.InputEvent.CTRL_MASK));
                exitMnuIt.setMnemonic('x');
                exitMnuIt.setText("Exit");
```

```
            actionMnu.add(exitMnuIt);
            menuBar.add(actionMnu);

            preferencesMnu.setText("Preferences");
            preferencesMnu.setMnemonic('p');
            manageAcctsMnuIt.setText("Manage A/cs");
            manageAcctsMnuIt.setMnemonic('m');
            manageAcctsMnuIt.setAccelerator(
                    javax.swing.KeyStroke.getKeyStroke(
                            java.awt.event.KeyEvent.VK_M,
                            java.awt.event.InputEvent.CTRL_MASK));
            preferencesMnu.add(manageAcctsMnuIt);
            configureMnuIt.setText("Configure");
            configureMnuIt.setMnemonic('c');
            configureMnuIt.setAccelerator(
                    javax.swing.KeyStroke.getKeyStroke(
                            java.awt.event.KeyEvent.VK_N,
                            java.awt.event.InputEvent.CTRL_MASK));
            preferencesMnu.add(configureMnuIt);
            menuBar.add(preferencesMnu);

            helpMnu.setMnemonic('H');
            helpMnu.setText("Help");

            helpMnuIt.setAccelerator(
                    javax.swing.KeyStroke.getKeyStroke(
                            java.awt.event.KeyEvent.VK_F1,
                            java.awt.event.InputEvent.CTRL_MASK));
            helpMnuIt.setMnemonic('h');
            helpMnuIt.setText("Help");
            helpMnu.add(helpMnuIt);

            visitSiteMnuIt.setAccelerator(
                    javax.swing.KeyStroke.getKeyStroke(
                            java.awt.event.KeyEvent.VK_V,
                            java.awt.event.InputEvent.CTRL_MASK));
            visitSiteMnuIt.setMnemonic('v');
            visitSiteMnuIt.setText("Visit JavaForU.com");
            helpMnu.add(visitSiteMnuIt);

            aboutMambaMnuIt.setAccelerator(
                    javax.swing.KeyStroke.getKeyStroke(
                            java.awt.event.KeyEvent.VK_B,
                            java.awt.event.InputEvent.CTRL_MASK));
            aboutMambaMnuIt.setMnemonic('b');
            aboutMambaMnuIt.setText("About BlackMamba");
            helpMnu.add(aboutMambaMnuIt);
            menuBar.add(helpMnu);
            setJMenuBar(menuBar);

            pack();
        } //GEN-END:initComponents
```

Actionların hazırlanması

prepareMenus ile, Actionlar tanımlanır.

```
        public void init()
        {
                mambaFrm = new MambaFrm();

                prepareMenus();

                mambaFrm.setExtendedState(JFrame.MAXIMIZED_BOTH);
                mambaFrm.mainPnl.add(new JLabel(Setup.getIcon()));
                mambaFrm.show();
        }
```

Buradaki kodda bir hata var. Actionlar, normalde eylemleri tanımlaması gerekiyorken, burada ActionListener ile eylemler tanımlanmış. Bu bir geliştirme hatası.

Dikkat ederseniz bazı actionlar, Runnable ile yürütülüyor, bazıları direk. Runnable olarak tanımladıklarımız, swing dışı bir zincirden çağrılan actionlar.

```
protected void prepareMenus()
{
        //Login
        ActionPropsSetter.setActionProps(Actions.loginAction, mambaFrm.loginMnuIt);
        mambaFrm.loginMnuIt.setAction(Actions.loginAction);
        mambaFrm.loginMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        login();
                }
        });

        //Logout
        ActionPropsSetter.setActionProps(Actions.logoutAction, mambaFrm.logoutMnuIt);
        mambaFrm.logoutMnuIt.setAction(Actions.logoutAction);
        mambaFrm.logoutMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        Runnable runnable = new Runnable()
                        {
                                public void run()
                                {
                                        logout();
                                }
                        };
                        SwingUtilities.invokeLater(runnable);
                }
        });

        //Exit
        mambaFrm.exitMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        exit();
                }
        });

        //Manage accts
        mambaFrm.manageAcctsMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        manageAccounts();
                }
        });

        //Configure
        mambaFrm.configureMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        configure();
                }
        });

        //Exit
        mambaFrm.setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
        mambaFrm.addWindowListener(new java.awt.event.WindowAdapter()
        {
                public void windowClosing(java.awt.event.WindowEvent evt)
                {
                        exit();
                }
        });

        //Help
        mambaFrm.helpMnuIt.addActionListener(new ActionListener()
        {
                String url = "";
```

```
                public void actionPerformed(ActionEvent ae)
                {
                        try
                        {
                                if (url.equals(""))
                                {
                                        url = Setup.getHelpFile().toURL().toExternalForm();
                                }

                                BrowserLauncher.openURL(url);
                        }
                        catch (IOException e)
                        {
                                if (Setup.DEBUG)
                                {
                                        e.printStackTrace();
                                }
                        }
                }
        });

        //Visit Site
        mambaFrm.visitSiteMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        try
                        {
                                BrowserLauncher.openURL("http://www.javaforu.com");
                        }
                        catch (IOException e)
                        {
                                if (Setup.DEBUG)
                                {
                                        e.printStackTrace();
                                }
                        }
                }
        });

        //About
        mambaFrm.aboutMambaMnuIt.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent ae)
                {
                        about();
                }
        });

        Actions.logoutAction.setEnabled(false);
}
```

Actionların işleri

```
protected void logout()
{
        if (beforeLogout[0] != null)
        {
                beforeLogout[0].call();
                beforeLogout[0] = null;
        }
}

protected void manageAccounts()
{
        final Callback onManageAccts = new Callback()
        {
                public void call()
                {
                }
```

```
        };

        Runnable runnable = new Runnable()
        {
                public void run()
                {
                        Accounts accounts = new Accounts();
                        accounts.init(mambaFrm, onManageAccts);
                }
        };
        SwingUtilities.invokeLater(runnable);
    }

    protected void configure()
    {
        Runnable runnable = new Runnable()
        {

                public void run()
                {
                        Configure configure = new Configure();
                        configure.init(mambaFrm);
                }
        };
        SwingUtilities.invokeLater(runnable);
    }

    protected void exit()
    {
        int option =
                JOptionPane.showConfirmDialog(
                        mambaFrm,
                        "Exit BlackMamba?",
                        "",
                        JOptionPane.YES_NO_OPTION);
        if (option != JOptionPane.YES_OPTION)
        {
                return;
        }

        Runnable logoutRunnable = new Runnable()
        {
                public void run()
                {
                        logout();
                        System.exit(0);
                }
        };
        SwingUtilities.invokeLater(logoutRunnable);
    }
```

about diyalog kutusu

```
    protected void about()
    {
        AboutDlg aboutDlg = new AboutDlg(mambaFrm, true);
        aboutDlg.contentLbl.setIcon(Setup.getIcon());
        aboutDlg.contentLbl.setText(ABOUT_STR);

        aboutDlg.pack();
        aboutDlg.setLocationRelativeTo(null);
        aboutDlg.show();
    }
```

Yukarıdaki kodda, yeni bir pencerenin nasıl oluşturulduğunu da görebiliriz.

```
public class AboutDlg extends javax.swing.JDialog ...
    public AboutDlg(java.awt.Frame parent, boolean modal)
    {
        super(parent, modal);
        initComponents();
    }
```

initComponents yine görsel nesnelerin yerleştirilmesini yapıyor.

En sonda yeni pencerenin gösterilmesi için:

```
aboutDlg.pack();
aboutDlg.setLocationRelativeTo(null);
aboutDlg.show();
```

login

```
protected void login()
{
        Runnable runnable = new Runnable()
        {
                public void run()
                {
                        Login login = new Login();
                        login.init(mambaFrm, beforeLogout);
                }
        };
        SwingUtilities.invokeLater(runnable);
}
```

Login bir kontrol sınıfı. Şu diyalog kutusunu çıkarıyor:

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\50.gif

```
public class Login
        public void init(MambaFrm frm, Callback[] beforeLogout)
        {
                mambaFrm = frm;
                this.beforeLogout = beforeLogout;

                loginDlg = new LoginDlg(mambaFrm, true);
                loginDlg.setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
                loginDlg.addWindowListener(new java.awt.event.WindowAdapter()
                {
                        public void windowClosing(java.awt.event.WindowEvent evt)
                        {
                                dispose();
                        }
                });

                comboBoxModel = new DefaultComboBoxModel();
                loginDlg.loginAliasesCbo.setModel(comboBoxModel);

                prepareComboBox();
                prepareButtons();

                loginDlg.setLocationRelativeTo(null);
                loginDlg.show();
        }

        protected void prepareComboBox()
        {
                Runnable runnable = new Runnable()
                {
                        public void run()
                        {
                                AccountsDatabase accountsDatabase = Setup.getAccountsDatabase();
                                comboBoxModel.removeAllElements();
                                Set set = null;
                                set = accountsDatabase.getAccountDetails();
                                for (Iterator iter = set.iterator(); iter.hasNext();)
                                {
                                        AccountDetails accountDetails = (AccountDetails) iter.next();
                                        comboBoxModel.addElement(accountDetails);
                                }
                                loginDlg.connectBtn.setEnabled(true);
                                loginDlg.loginAliasesCbo.setEnabled(true);
```

```
                        if (set.size() == 0)
                        {
                                loginDlg.connectBtn.setEnabled(false);
                                loginDlg.loginAliasesCbo.setEnabled(false);
                                comboBoxModel.addElement(MSG_CREATE_ACCT);
                        }
                }
        };
        SwingUtilities.invokeLater(runnable);
}

protected void prepareButtons()
{
        //Manage a/cs
        loginDlg.manageBtn.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent e)
                {
                        manageAccounts();
                }
        });

        //Connect
        loginDlg.connectBtn.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent e)
                {
                        connect();
                }
        });

        //Cancel
        loginDlg.cancelBtn.addActionListener(new ActionListener()
        {
                public void actionPerformed(ActionEvent e)
                {
                        dispose();
                }
        });
}
```

manageAccounts

```
protected void manageAccounts()
{
        Runnable runnable = new Runnable()
        {
                public void run()
                {
                        Accounts accounts = new Accounts();
                        accounts.init(mambaFrm, onManageAccts);
                }
        };
        SwingUtilities.invokeLater(runnable);
}
```

Burada Accounts bir kontrol nesnesi. Şu diyalog kutusunu açıyor:

file:///F:\Belgelerim\Notlarim\Java%20Notlarim\Resimler\Makaleler\49.gif

Bunu bir callback nesnesiyle ilklendiriyoruz. Böylece, yeni hesap eklendiği vakit, combo kutusunu yeniden güncelliyoruz.

```
private final Callback onManageAccts = new Callback()
{
        public void call()
        {
                prepareComboBox();
        }
};
```

prepareComboBoxta, diyalog kutusundan okunan verilerin kaydedildiği AccountsDatabasedeki hesaplar combo kutusuna ekleniyor.

```
	protected void prepareComboBox()
	{
		Runnable runnable = new Runnable()
		{
			public void run()
			{
				AccountsDatabase accountsDatabase = Setup.getAccountsDatabase();
				comboBoxModel.removeAllElements();
				Set set = null;
				set = accountsDatabase.getAccountDetails();
				for (Iterator iter = set.iterator(); iter.hasNext();)
				{
					AccountDetails accountDetails = (AccountDetails) iter.next();
					comboBoxModel.addElement(accountDetails);
				}
				loginDlg.connectBtn.setEnabled(true);
				loginDlg.loginAliasesCbo.setEnabled(true);

				if (set.size() == 0)
				{
					loginDlg.connectBtn.setEnabled(false);
					loginDlg.loginAliasesCbo.setEnabled(false);
					comboBoxModel.addElement(MSG_CREATE_ACCT);
				}
			}
		};
		SwingUtilities.invokeLater(runnable);
	}
```

Peki bu callback ne zaman çağrılıyor?

Yukarıdaki diyalog kutusunda Save düğmesine basıldığında:

Yukarıdaki diyalog kutusunun yöneticisi (Accounts) bir ListConfig´dir. Bütün ListConfigler aynı mekanizmayla çalışıyor: Solda bir JList. Ortada çeşitli veri toplama cihazları (widget). Sağda Kaydet, Ekle düğmeleri...

```
public abstract class ListConfig...
		listConfigPnl.saveBtn.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				Object newObj = saveBtnClicked();

				if (newObj == null)
				{
					return;
				}

				//Save existing
				listModel.removeElement(newObj);
				int sel = listModel.addElement(newObj);
				listConfigPnl.aliasesLst.getSelectionModel().setSelectionInterval(sel,
sel);
				listConfigPnl.aliasesLst.ensureIndexIsVisible(sel);
			}
		});
```

Yukarıdaki metotta, bütün kaydet komutlarının ortak işi yapılıyor. Soldaki listeye eklenen yeni nesne (burada bir AccountDetails) konuluyor. Burada döndürülen nesne seçili hale getiriliyor. ensureIndexIsVisible ile, listenin içinde seçili nesnenin görünür kılınması temin ediliyor (scroll etmek gerekirse).

saveBtnClicked, alt sınıflarda somutlaştırılıyor. Bu her somut diyalog kutusunun veri toplama ihtiyacını karşılıyor. Her birinde çünkü farklı alanlar var:

```
	protected Object saveBtnClicked()
	{
		AccountDetails accountDetails = null;

		if (manageAcctsPnl.aliasTxt.getText().equals(""))
		{
			manageAcctsPnl.aliasTxt.requestFocus();
```

```
                return accountDetails;
            }
            if (manageAcctsPnl.serverTxt.getText().equals(""))
            {
                manageAcctsPnl.serverTxt.requestFocus();
                return accountDetails;
            }
            if (manageAcctsPnl.portTxt.getText().equals(""))
            {
                manageAcctsPnl.portTxt.requestFocus();
                return accountDetails;
            }
            if (manageAcctsPnl.loginTxt.getText().equals(""))
            {
                manageAcctsPnl.loginTxt.requestFocus();
                return accountDetails;
            }
            String password = new String(manageAcctsPnl.passwordTxt.getPassword());
            if (password.equals(""))
            {
                manageAcctsPnl.passwordTxt.requestFocus();
                return accountDetails;
            }

            accountDetails = new AccountDetails();
            accountDetails.setAlias(manageAcctsPnl.aliasTxt.getText());
            accountDetails.setServerAddress(manageAcctsPnl.serverTxt.getText());
            accountDetails.setServerPort(Integer.parseInt(manageAcctsPnl.portTxt.getText()));
            accountDetails.setLogin(manageAcctsPnl.loginTxt.getText());
            accountDetails.setPassword(password);

            return accountDetails;
        }
```

Eğer alanlar boş bırakılmışsa, swing onların üzerine imleci getiriyor (requestFocus).

Iş akışını nasıl göstermeli?

Az sayıda işlev olmasına rağmen, sunum-uygulama akışı-iş modeli katmanları birbirinden ayrılmış olmasına rağmen, mantığı kavramak yine de zor...

Nasıl yapmalı, nasıl daha da kolaylaştırmalı?

Bir çözüm, kullanıcı arayüzlerinin sadeleştirilmiş şekilde kağıt üzerinde çizip, buradaki düğmelere basılınca hangi hareketlerin oluşacağını göstermek olabilir.

```
Login - Diyalog Kutusu
        Manage Button
                -» Accounts

Accounts - Diyalog Kutusu
        Close Button
                -» kaydet
                -» callback
                        -» prepareComboBox
```

Ancak böyle onlarca fonksiyon olması durumunda da bu doküman yine karmaşıklaşacak. Dolayısıyla, her ne kadar en azından okundukça anlaşılabilir ve sınırlar iyi çizilmiş olsa da, anlamak zaman gerektirecek.

Kontrol sınıfları

Ana ekran ve sıçrama ekranı da dahil olmak üzere bütün kullanıcı arayüzü, kontrol nesneleri tarafından oluşturuluyor.

Üçe ayırıyoruz uygulamayı. Model-sunum-kontrol. Sunum, sadece