İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

"JAVA BEAN"LERİ VE BİR VERİTABANINA ERİŞİM UYGULAMASI

YÜKSEK LİSANS TEZİ
Müh. E. Esra ATALAY
(Enstitü No: 504950610011)

83017

Tezin Enstitüye Verildiği Tarih : 11 Ocak 1999
Tezin Savunulduğu Tarih : 05 Şubat 1999

Tez Danışmanı :	Prof.Dr. A. Emre HARMANCI	12.04.99
Diğer Jüri Üyeleri	Doç.Dr. Nadia ERDOĞAN	
	Doç.Dr. Tevfik AKGÜN	12.4.99

ŞUBAT 1999

ÖNSÖZ

Bana çalışma konumu belirlememde ve daha sonraki çalışmalarımda destek veren ve her türlü yardımı sağlayan değerli hocam Prof. Dr. A. Emre HARMANCI'ya ve yine bu tez çalışmam süresince bana her türlü desteği sağlayan eşim Fatih USTA'ya ve benim yanımda yer alan İTÜ Bilgi İşlem Merkezi'ndeki tüm arkadaşlarıma teşekkür ederim.

E. Esra ATALAY

Ocak 1999

İÇİNDEKİLER

ŞEKİL LİSTESİ

"JAVA BEAN"LERİ VE BİR VERİTABANINA ERİŞİM UYGULAMASI

ÖZET

Günümüzde yazılım teknolojisi, tekrar kullanılabilir yazılım birimlerini tanımlamak yönünde çalışmalarla ilerlemektedir. Fonksiyon kütüphanelerinden ve nesnelerden sonra, bugün, yazılım bileşenleri en yaygın teknoloji haline gelmiştir. Son dönem yazılım bileşenleri, görsel araçlar aracılığıyla yerel uyarlamayı ve birkaçının, kod içinde birlikte kullanılmasını sağlamaktadır. Bu sayede uygulama geliştiriciler, ihtiyaç duydukları bileşeni kendi uygulamarına uygun hale getirmek için yerel uyarlarlar ve ek kod yazmaya gerek duymazlar. Ancak çoğu bileşen yapısı, belirli bir donanıma ve işletim sistemine bağlı kaldığı için, günümüzde gelişen internet ve intranet uygulamalarında kolayca kullanılamamaktadır. Ayrıca ağ üzerinden kullanımda güvenlik açısından problemler ortaya çıkmaktadır. JavaBean'ler ise, bu sakıncaları içermedikleri ve Java'nın platformdan bağımsız ve güvenli olma özelliğini temel aldıkları için, hızla yaygınlaşmaktadır.

Bu çalışmada, İTÜ Proje Yönetim Merkezi için, İTÜ Bilgi İşlem Merkezi tarafından geliştirilen bir otomasyon yazılımının bir bölümü gerçekleştirilmektedir. Bu çalışma, bir veritabanında tutulan, projelere ait süre ve maliyet bilgilerine erişimi temel alır. Amaç, kullanıcının web üzerinden bu bilgilerin grafiksel sunumuna erişiminin ve belirli kriterlerle farklı gösterimler elde ederek, projelere ayrılacak maliyetler konusunda karar vermesinin sağlanmasıdır. Bu tanımı ile bu çalışma, interaktif bir uygulama olduğundan ve web üzerinden erişim temel alındığından, Java kavramlarının kullanımına yol açar. Bu problemin çözüm yöntemi olarak JavaBean'ler seçilmiştir.

"JAVA BEANS" AND A DATABASE ACCESS APPLICATION

SUMMARY

Recently, software technology has improved with the studies on reusable software units. After function libraries and objects, software components have become the most widespread and preferred technology. The recent-time software components support customization by visual tools and use in a software code. So, application developers customize the component they need for their application and do not have to write additional code. However, because most component architectures stay dependent on a particular hardware and operating system, they can not keep up with internet and intranet applications. Besides, their use over web brings security problems. As JavaBeans do not have these negative properties and rely on the platform independent and secure nature of Java, they get widespread and known.

In this study, a part of an otomation software that is being developed by İTÜ Computer Center for İTÜ Project Management Center has been implemented. The basis of this study is the access to the timing and cost information of the projects kept in a database. The goal is to let the user access the graphical view and having different views according to particular cost kriteria, decide on the cost that will be saved for projects. With this definition, this study causes the use of Java because it is an interactive application and has the basis of web access. For this reason, this study solves the problem with JavaBeans.

1. GİRİŞ

1.1 JavaBean'lerin Tanımı

JavaBean'ler, ilk olarak 1996 yılında tanımlanmıştır. Bean API (Application Programmer Interface) amacının ilk ve dar tanımı şöyle verilmiştir :

"Bir Bean, görsel olarak bir tasarım aracında düzenlenebilen, tekrar kullanılabilir bir yazılım bileşenidir." [1]

Java Bean'ler, bu API amacını gerçeklemektedirler ve hatta bu belirtilen tanımın dışında da bazı belirleyici özellikleri vardır. Bu yüzden, tekrar kullanılabilir ve uygulama geliştirme araçları tarafından düzenlenebilir olmalarını belirten bu tanımdan daha geniş bir tanıma ihtiyaç duyulmaktadır. O halde, Java Bean'lerin daha doğru bir tanımı şu şekilde yapılabilir :

"Bir Bean, bir uygulama geliştirme aracında, canlı bir nesne olarak düzenlenebilen bir bileşendir. Bean, kendisini, yerel uyarlayıcılarını, özelliklerini, özellik düzenleyicilerini, olaylarını ve metodlarını tanımlayan bir BeanInfo nesnesi ile ilişkilidir." [2]

1.2 Yazılım Bileşen Yapıları

Bir yazılım bileşeni, tekrar kullanılabilen bir yazılım yapı taşıdır. Bir yazılım bileşen yapısı, mevcut yazılım bileşenlerini birleştirerek uygulama geliştirmeyi mümkün kılar. Bir taşıyıcı, bir veya daha fazla bileşeni birlikte gruplar, böylece bileşenlerin taşıyıcıyla olduğu gibi, kendi aralarında da iletişim kurabilmelerini sağlar. [3]

1.3 Bileşen Modelleri İle İlgili Bazı Kavramlar

Bir bileşen modeli tarafından tanımlanan yapı, öncelikle, bileşenlerin dinamik bir çevrede nasıl ilişkilendirileceklerine karar vermekle yükümlüdür. Yazılım bileşenlerinin yaygın kullanımına bakılırsa, bileşen modellerini ve ilişkili yapılarını anlamanın önemi bir kez daha anlaşılmış olur. [4]

Tüm yazılım bileşen modelleri, iki temel eleman tanımlarlar: bileşenler ve taşıyıcılar. Bir bileşen modelinin bileşen kısmı, farklı bileşenlerin nasıl yaratıldığı ve kullanıldığını belirler. Diğer bir deyişle, bileşen modeli, bileşenlerin oluşturulduğu şablonu sağlar. Taşıyıcı kısmı ise, bileşenleri, verimli yapılarda birleştirmenin yolunu tanımlar. İçerenler, kendileri de bileşenler olabilirler. Bu sayede, bileşenler birbirlerinin içinde yer alacak şekilde yerleştirilerek karmaşık arayüzler oluşturulabilir.

Bir bileşen modeli, bazı servisleri sağlamakla sorumludur. Bunlar arasında başlıca beş tanesi şunlardır:

- İçgözlem

- Olay işleme

- Kalıcılık

- Uygulama geliştirici desteği

- Dağıtılmış işleme desteği

1.4 Yazılım Bileşenlerinin Bir Ide (Integrated Application Development Environment) İçindeki Tipik Davranışı

Bir IDE içinde, yazılım bileşenlerinin başlıca üç özelliği vardır: özellikler, olaylar ve metodlar.

1.4.1 Bileşen Özellikleri

Özellikler, bir bileşenin nitelikleridir. Genellikle, bir IDE içinde, bir özellik düzenleyici kullanılarak düzenlenebilirler. Bir özellik düzenleyici, geliştiricinin, bir bileşenin özelliklerini görmesine ve değiştirmesine izin verir. Bu, bileşen yapısının, bir bileşenin özelliklerini, IDE'ye açması için, bir mekanizmaya sahip olmasını gerektirir.

Bir uygulama geliştiren kişi, uygulamasını sakladığında, bileşenlerin özelliklerinin durumları da saklanır. Buna `kalıcılık` denir. Uygulama çalıştırıldığında, bileşenlerin özellikleri, saklanmış değerlerini alırlar. Uygulamanın kendisi de, bir bileşenin özelliklerini okuyabilir veya değiştirebilir.

1.4.2 Bileşen Olayları

Çoğu bileşenler, kendilerine olacak bazı olaylara karşı yapılacak işlemler içerirler. Geliştirici kişi, bir olayın sonucunda ne yapılması gerektiğini, belirli bir bileşenin olayıyla, ilgili bir kod bölümünü ilişkilendirerek belirler. Çoğu IDE, geliştirici, kodla ilişkilendirecek bileşen/olay çiftini belirlediğinde görünen bir kod penceresi içerir.

1.4.3 Bileşen Metodları

Bir bileşenin metodları, uygulamadan veya IDE'den çağırılabilen fonksiyonlardır. Metodlar genellikle, bir bileşenin özelliklerini okumak ve ayarlamak için kullanılırlar.

1.5 Başlıca Bileşen Yapıları

Günümüzde, Java Bean'lerden başka, bir çok bileşen yapısı vardır ve bunlardan iyi bilinen bir kaçı aşağıda incelenmiştir :

1.5.1 Netscape'den LiveConnect

LiveConnect, Netscape'in web inceleyicilerinde kullanılan ve bileşenler arasında iletişime izin veren bir yapıdır. Netscape inceleyicilerinde koşan üç temel bileşen tipi vardır:

- Java appletleri

- JavaScript kodu ve ilişkili nesneler

- Netscape plug-inleri

LiveConnect, bu üç tip bileşenin birbiriyle haberleşmesine izin verir. LiveConnect ile, web sayfası bir taşıyıcı haline gelir. Birbiriyle ve taşıyıcıyla haberleşebilen bileşenler de onun içinde yer alırlar. Bu yapı sayesinde, web-tabanlı bir uygulama, bir browser penceresinde yer alabilir ve çerçevelerden, Java appletlerden, plug-inlerden ve JavaScript kodu tarafından denetlenen HTML'den oluşabilir.

1.5.2 CI Labs'dan OpenDoc

OpenDoc, platformlar-arası bir yazılım bileşen yapısıdır. OpenDoc, Bileşen Yapısı üzerinden GUI (Graphical User Interface_Grafik Kullanıcı Arayüzü) sihirbazları ve Döküman Yapısı üzerinden de gömülü dökümanlar gibi yazılım bileşenlerini destekler. Bu standartlara göre geliştirilen bileşenler, beraber çalışabilirler.

OpenDoc, OpenDoc bileşenlerini bir yazı dili kullanarak denetlemeyi sağlayan, Open Scripting Architecture adında bir API'ya sahiptir.

1.5.3 Microsoft'tan ActiveX

ActiveX, özellikle web için optimize edilmiştir. Adını da, web sayfalarına, yürütülebilir (aktif) içerik kazandırmasından almıştır. ActiveX bileşenleri, `kontroller` olarak da isimlendirilirler ve Microsoft'un ve diğer ActiveX kontrol satıcılarının da sağladığı bir çok kontrol mevcuttur. Visual Basic, Visual C++ ve Visual J++ , pek çok dilde geliştirilebilen ActiveX kontrollerini desteklerler.

ActiveX'in altındaki ana teknoloji Microsoft'tan DCOM'dur (Distributed Component Object Model – Dağıtılmış Bileşen Nesne Modeli). DCOM, ActiveX bileşenlerinin haberleşmesi için gerekli altyapıyı sağlar.

2. JAVABEAN BİLEŞENİNE GENEL BAKIŞ

2.1 Javabean'ler ve Özellikleri

2.1.1 Javasoft'un Tanımladığı Görev

JavaSoft'un, Java ile birlikte gitmesini planladığı bileşen teknolojisi olan JavaBean kavramı ve özellikleri, JavaSoft'un kendi JavaBeans görev tanımı ile verilebilir : 'Birkez yaz, her yerde koştur, her yerde kullan.'

2.1.1.1 Birkez Yaz

Yazılım geliştiriciler, uygulamalarında değişiklik yapmak istediklerinde, çoğu zaman kodlarını tekrardan yazmak zorunda kalırlar. Bunun başlıca sebebi, platform değişikliklerinde tamamen farklı kodlara ihtiyaç duyulmasıdır. JavaSoft ise, iyi geliştirilmiş bir yazılım bileşeni teknolojisinin, kodun bir kez yazılmasını ve eklemeler ve geliştirmeler içinse, tekrardan yazmalara ihtiyaç duymamayı sağlamasının gerekli olduğunu savunmaktadır. Buna dayandırılırsa, JavaBeans, mevcut bir kodu geliştirmek için, orjinal kod üzerinde tekrar çalışmayı gerektirmeyen yollar sağlamalıdır.

JavaBean bileşenlerini birkez yazmak, sürüm denetimi için de önemlidir. Böylece bileşen geliştirenler, belli bölümleri tekrar yazmak yerine, sadece bileşenlere artan değişiklikler yapabilirler. Bu da işlevselliğin düzenli artışı, aynı zamanda bir bileşenin artan sürümler ile gelişimini getirir.

2.1.1.2 Her Yerde Koştur

Bu özellik, JavaBean bileşenlerinin her ortamda çalıştırılabilir olmasına işaret eder. Yani, JavaBean'lerinin platformdan bağımsız olmasının gerekliliğinden bahseder. Bugünün yazılım şartlarında, bir bileşenin amacına ulaşıp, başarılı olabilmesi için bu

özelliğin sağlanması gereklidir. JavaBean'ler için ise, platformdan bağımsız olma desteği, kolaylıkla sağlanabilir, çünkü Java dili bu temele dayanır.

"Her yerde koştur" ifadesi, JavaBean'lerinin sadece farklı platformlarda değil, dağıtılmış ağ ortamlarında da yürütülebilirliğine işaret eder.

2.1.1.3 Her Yerde Kullan

Bu özellik, JavaBean bileşenlerinin, değişik yapılarda, değişik amaçlar için kullanılabilirliğini işaret eder. JavaBean'lerinin uygulamalar, diğer bileşenler, dökümanlar, Web site'leri ve uygulama geliştirme araçları da dahil olmak üzere pek çok ortamda tekrar kullanılabilirliklerini savunur. Bu gereksinim de, yazılım bileşenlerinin birincil amacı olan, kodun tekrar kullanılmasını sağlar.

2.1.2 JavaBean'lerin Tasarım Amacını Gerçekleştirmeleri

JavaBean'lerin başlıca tasarım amaçları, JavaBean bileşenleri için aşağıda verilen gereksinimler listesiyle özetlenebilir:

- Küçük, kod yoğunluğu yüksek ve oluşturulması ve kullanımı kolay olmalı

- Tamamen taşınabilir olmalı

- Java'ya özgü güçlü özellikleri temel almalı

- Tasarım-zamanı, esnek bileşen editörlerini desteklemeli

- Gürbüz, dağıtılmış bilgi işleme mekanizmalarını desteklemeli

JavaBean'lerin bu özellikleri, bu bileşenlerin çoğu zaman, düşük hızlı Internet bağlantıları üzerinden aktarılacakları dağıtılmış ortamlarda kullanılacakları için önemlidir. Açıkça, bileşenler, aktarım zamanını kabul edilebilir bir düzeyde tutabilmek için, mümkün oldukça küçük olmalıdırlar. Bu amacın ikinci bölümü, bileşenlerin oluşturulması ve kullanımı sırasındaki kolaylıkla ilgilidir. Bileşenlerin kullanımının kolay olması ve bileşen oluşturmayı kolaylaştıran bir bileşen yapısının oluşturulması farklı olaylardır. Mevcut bileşen yazılımları, bileşen geliştirenleri zora sokan, karmaşık programlama API'leri getirmişlerdir. Bu yüzden, JavaBean

bileşenleri, sadece kullanımı kolay değil, aynı zamanda geliştirilmesi de kolay olmalıdırlar.

JavaBean'ler, geleneksel Java applet programlamada kullanılan sınıf yapısına bağlı oldukları için, Java uygulaması geliştiren kişiler için büyük kolaylık sağlarlar. Bu, JavaBean'leri küçük tutmakta fayda sağlar, çünkü Java dilinin kod yoğunluğu yüksektir.

2.1.2.1 Taşınabilir

JavaSoft'un JavaBean'leri yaratmaktaki en büyük amaçlarından biri, tamamen taşınabilir olmalarıydı. JavaBean API'si, platform-bağımsız Java sistemi ile birlikte, platform-bağımsız bileşen çözümünü getirmektedir. Bir bileşen geliştirmek ve onu değiştirmeden Java-destekleyen her sistemde çalıştırmak mümkündür.

2.1.2.2 Java'nın Güçlü Özelliklerine Dayanır

Mevcut Java mimarisi, bileşenlere kolaylıkla uygulanabilen bir çok fayda sağlar. Bunların önemlilerinden biri, nesnelerin çalışma sırasında birbirleriyle dinamik olarak iletişim kurmalarını sağlayan, sınıf keşif mekanizmasıdır. Bu, nesnelerin gelişim geçmişlerinden bağımsız olarak birleştirilebileceklerine işaret eder.

JavaBean'lerin mevcut Java işlevselliğinden miras almalarına bir başka örnek, bir nesnenin durumunu saklaması ve tekrar alması demek olan "kalıcılık"tır. Kalıcılık, Java'da bulunan "serileştirme" (serialization) mekanizması kullanılarak, JavaBean'lerde otomatik olarak gerçekleştirilmektedir.

2.1.2.3 Uygulama Geliştirme Desteği

JavaBean'lerin bir başka tasarım amacı, tasarım-zamanı konuları ve uygulama geliştiricilerin JavaBean bileşenlerini kullanma şekilleridir. JavaBean yapısı, tasarım-zamanı özelliklerinin belirlenmesi ve bileşenlerin görsel olarak düzenlenmesini sağlayacak düzenleme mekanizmaları için destek içerir. Sonuç olarak, geliştiriciler, görsel uygulama geliştirme araçlarını kullanarak, JavaBean bileşenlerini birleştirebilir ve değiştirebilirler. Bu özellik, tek başına, profesyonel uygulama

geliştirme araçlarının kullanımına imkan tanır ve uygulama geliştiricilerin verimliliğini artırır.

2.1.2.4 Dağıtılmış Bilgi İşleme Desteği

Dağıtılmış bilgi işleme için verilen destek, JavaBean yapısının çekirdek elemanı değilse de, JavaBean'lerin önemli bir özelliğini oluşturur. Bunun nedeni, dağıtılmış bilgi işlemenin karmaşık çözümler gerektirmesidir. Bu sebeple, JavaBean, gerekli olduğunda dağıtılmış bilgi işleme mekanizmalarının kullanılmasına izin verir, ama çekirdek destek vererek de kendini fazla yüklemez.

JavaBean bileşeni geliştirenler, kendi ihtiyaçlarına en iyi şekilde uyacak dağıtılmış bilgi işleme yaklaşımını seçebilirler. JavaSoft, kendi dağıtılmış bilgi işleme çözümünü, Java Remote Method Invocation (RMI) teknolojisinde, sağlar ama JavaBean geliştirenler bu çözümle kısıtlı değildirler. Diğer seçenekler arasında, CORBA ve Microsoft'un DCOM'u sayılabilir. Önemli nokta, dağıtılmış bilgi işlemenin, bilinçli olarak JavaBean'lerin dışında bırakılmış olması, ama isteyenlere destek veriliyor olmasıdır.

2.2 Diğer Bileşen Modelleri ve JavaBean'ler

JavaBean'lerin yukarıda belirttiğimiz özelliklerinin bir bölümü, şu anda mevcut diğer bileşen modelleri tarafından desteklenmemektedir.

2.2.1 İşlevsellik Yönünden Karşılaştırma

Bir yazılım bileşeninin tekrar kullanılabilirliğini kısıtlayacak en büyük etken, kullanılmakta olan farklı donanım yapıları ve işletim sistemleridir. Bileşenlerin geliştirilmesi ile ilgili pek çok çalışma yapılmış ve ürünler çıkarılmıştır ama hepsi bir tek işletim sistemine bağımlı kalmıştır. Microsoft'un geliştirdiği bazı bileşen yapılarını incelersek (VBX ve OCX), bunların PC pazarında ve Windows işletim sisteminde geçerli teknolojiler olduklarını görürüz. Özellikle Internet'in kullanımının yaygınlaşması ile, kullanılan her türlü yazılımın platformdan bağımsız çalışabilirliği önem kazanmıştır. Platform bağımlılığının yanısıra, geliştirme ortamı ve

programlama diline olan bağımlılık da bileşenlerin herkes tarafından rahatlıkla kullanılmasının önünde bir engel oluşturmaktadır.

Microsoft'un ActiveX ve OpenDoc gibi mevcut bileşen modellerini incelersek, bazı problemler içerdiklerini görürüz:

- Bileşen, C veya C++'da yazılmışsa, platforma bağımlı bir çalışabilir dosya oluşturmak üzere derlenmiştir. Bu sebeple, işletim sistemine özel çağrılar kullanır.

- Bir bileşen için çok büyük miktarlarda kod yazmak gereklidir.

- Mevcut çatılar kullanılsa bile, yine de bileşen yazmak oldukça zor bir iştir.

Bahsettiğimiz bu özellikler, bu olumsuzluklara sahip olmayan ve platformdan bağımsız bir yapıya sahip olan, Java'nın bileşen teknolojisi, JavaBean'lerin tercih edilmesinin başlıca sebebidir. Yukarıda bahsettiğimiz sorunlara Java'nın getirdiği çözümleri incelersek:

- JavaBean'ler, Java'da yazılırlar ve platforma özel kod içermezler.

- Basit veya oldukça karmaşık bileşenleri geliştirmek mümkündür. Çok büyük bileşenler elde edilmez.

- Bir Java bileşeni geliştirmek, diğer programlama dilleriyle kıyaslandığında kolaydır.

- Yazılan bir JavaBean'i, ActiveX veya OpenDoc gibi diğer bileşen modellerinde de, kullanmak mümkündür ve doğal bir bileşen gibi davranacaktır. Tüm iletişim ve diyalog, bir "köprü" tarafından sağlanacaktır.

- Bir Java bileşeni, standart dağıtılmış nesne (JavaIDL veya RMI_Remote Method Invocation) veya dağıtılmış veri (JDBC) mekanizmalarından birini, uzaktaki veriye ulaşmak için kullanabilir.

Visual Basic, Visual C++ ve Delphi gibi gelişmiş görsel araçların kullandıkları bileşenler, pek çok yönde JavaBean'lerden farklılık gösterirler. Bu teknolojiler, bileşenlerinin oldukça statik ve sınırlı bir görüntüsüne dayanırlar. Bu araçları kullanmak, sonuçta, IDE tarafından kısmi kodun yazılmasını ve GUI (Graphical User

Interface – Grafik Kullanıcı Arayüzü) kurucu tarafından tanım dosyalarının yaratılmasını getirir.

Dosya-temelli bir tanımlamanın getireceği olumsuzluk, API'nin koddan ayrı olarak tanımlanmasının gerekli olmasıdır. Bu dosyalar da çoğunlukla, dille standartlaşmış olmanın aksine, firmaya özeldir. API tanımlama dosyalarını oluşturmak kolay değildir. Çünkü programcı tarafından ikinci bir programlama dili gibi öğrenilmeleri gereklidir. Bileşen tarif dosyalarının tanımı, genellikle, üretenin IDE'sine özeldir. Bu yüzden, kullanıcıların yeni bileşen oluşturmaları ve IDE'ye eklemeleri çok zordur. Yine aynı sebepten, farklı üreticilerin bileşenlerini paylaşmak da imkansızlaşır.

Çekirdekten Yansıma API'si (Core Reflection API) ve JavaBean API'si, yazılımın programlanabilir görüntüsünü sağlar. Bileşenler, sadece programın nasıl görünmesi gerektiğini gösteren bir GUI düzenleyicinin aksine, doğrudan çalışan yazılımın içinde düzenlenirler. Yani, bir program, Bean arayüzünü, Bean'in sınıf dosyasını okuyarak anlayabilir. Bu da, bir Bean'in metodlarının IDE araçları tarafından araştırılabileceğini ve geliştiricinin, koşma anında, Bean'in davranışını görebileceğini gösterir. Bean'lerin görünebilirliği ve düzenlenebilirliği, Java Çekirdekten Yansıma API'si ile mümkündür. Çekirdekten Yansıma, geliştiricilerin, diğer programları analiz edebilen ve yorumlayabilen kod yazmalarına imkan verir. Bu sebeple, Bean'ler, ek tanımlama dosyalarına veya ek sınıflara ihtiyaç duymazlar. Java diline yapılan bir geliştirme sonucunda tanımlanan Çekirdekten Yansıma kullanılarak, Bean, kod seviyesinde incelenebilir.

Çekirdekten Yansıma, sınıflar ve arayüzler ile, onlara karşılık düşen metodlar ve alanların incelenmesidir. Bean'lerin ortamında, İçgözlem sınıfı, Çekirdekten Yansıma API'sini kullanarak, bean'e özellikleri, olayları ve metodlarını sorar. Sonuç olarak bean'in bileşen API'si olan bir BeanInfo nesnesi elde edilir. Bu aşamadan sonra, IDE, kullanıcıya bilgi görüntülemek, Bean'i düzenlemek ve kullanıcının geliştirdiği ana programla bütünleştirmek için BeanInfo nesnesini kullanabilir.

2.2.2 Güvenlik Yönünden Karşılaştırma

Bileşenlerle ilgili güvenlik konuları, bir web sayfasına erişim sonucunda bileşenlerin otomatik olarak yüklenebilirliğinden sonra ortaya çıkmıştır. Bu konuda en yaygın iki teknoloji, Java Bean'ler ve ActiveX'tir. Bu iki teknoloji de, güvenlik konusunu tamamen farklı şekillerde ele alırlar. JavaBean'ler, Java Görüntü Makinasında (Java Virtual Machine) koşarlar. Java'da yazılmayan ActiveX kontrolleri, Görüntü Makina'da koşmazlar.

2.2.2.1 Sandbox'da Güvenlik

Java ile birlikte, bir Java programına ev sahipliği yapan bir makinayı korumak için, bir ortam da geliştirilmiştir. Bu ortam, sandbox olarak isimlendilirmiştir ve aşağıdaki elemanlardan oluşmuştur :

- Java Virtual Machine (VM – Görüntü Makina) : Java VM, Java sınıflarını meydana getiren byte-kodu yorumlar ve çalıştırır.

- Byte-kod Doğrulayıcı : Byte-kod doğrulayıcı, byte-kod'un yasal işlemler gerçekleştireceğini ve Java VM'e zarar vermeyeceğini doğrular.

- Sınıf Yükleyici : Java sınıflarını yüklemekle sorumludur. Ayrıca Java sistem sınıflarının da doğru yerden yüklendiğini doğrular.

- Güvenlik Yöneticileri : Bir Java güvenlik yöneticisi, bir Java sınıfının sistem kaynaklarına erişimini denetler. Java sınıfları, appletler ve bean'ler dahil olmak üzere, varsayım olarak, bir web sayfasından yüklendiklerinde güvenilmez kabul edilirler. Bu yüzden, Java sınıflarının ev sahibi makinaya erişimleri oldukça kısıtlıdır. Örneğin, güvenilmeyen bir applet, bir dosyadan okuma veya dosyaya yazma yapamaz, yazıcıya yazı bastıramaz, getirildiği makinadan başka makina ile haberleşemez.

Java sınıfları, ev sahibi makinanın güvenini, güvenilir biri tarafından dijital olarak imzalandıklarında kazanırlar. Güvenilir sınıflar, dosya sistemine veya yazıcılara erişim hakkı kazanabilirler.

Tüm bu güvenlik önlemleri sayesinde, ev sahibi makina, ağdan yüklenen Java Bean'lere karşı güvendedir.

2.2.2.2 ActiveX'de Güvenlik

Java'da yazılmayan ActiveX bileşenleri, sandbox'ta koşmazlar. ActiveX dünyasında, byte-kod doğrulayıcının sağladığı güvenliğe karşı düşen bir kavram yoktur. ActiveX bileşenleri de dijital olarak imzalanabilirler ama bileşen tahmin edildiği gibi güvenli değilse, sistem kaynaklarını koruyacak güvenlik yöneticileri yoktur. Ayrıca, sandbox, (Java VM aracılığıyla) tutarlı bir yürütme ortamı sağladığından, her yerde hiçbir düzenlemeye gereksinim olmadan çalışacaktır. ActiveX kontrolleri ise, hedef platformun API'sine göre yazıldığından, tamamen platformlar arası taşınabilir değildirler.

2.3 JavaBean'lere Özel Bazı Temel Teknolojilerin Tanımlanması

Çoğu görsel araçta oluşturulan bileşenler, tasarım aşamasında, nesnelerin kendisi değil, sadece bir gösterimdir. Böyle bir model, karmaşık bileşenlerin gösterimini engeller, çünkü bir gösterim de tasarlanmalıdır.

2.3.1 Serileştirme

Java, bir nesne ile ilgili bilgileri, bir zincire kaydettiği bir nesne ağacı üzerinden işler. Bu zincir, bir dosya veya bir saklama birimi olabilir. Program tekrar başlatıldığında, zincir yeniden okunabilir ve sürekli bilgiyi içeren baştaki nesne ağacı yeniden oluşturulabilir. Bir nesneyi, bir zincire yazma veya zincirden okuma sürecine serileştirme denir.

Java Bean'lerin canlı nesneler olarak düzenlenmesinin bu kadar kolay olmasının başlıca sebebi, bir uygulama geliştirme aracı kullanan bir programcının değiştirebileceği ayarların kopyalarını oluşturmak için kod yazmanın gerekli olmamasıdır. Kod üretiminin gerekli olmamasının nedeni, Bean nesnesinin durumunun her an saklanabilir olmasıdır. Geliştiriciler ve IDE üreticileri, Bean'in kalıcı durumunu saklayarak, bileşenlerin oluşturulması ve konfigüre edilmesi

sırasında ortaya çıkacak ağır işin çoğundan kurtulabilirler. Bir Bean, saklanmış durumu geriserileştirilerek tekrardan oluşturulabilir.

2.3.2 Çekirdekten Yansıma ve İçgözlem

JavaBean API'sinin anahtar özelliklerinden biri, bir bean'i koşar durumdayken düzenleyebilmektir. Bunun olmasına izin veren teknoloji, Çekirdekten Yansıma API'sinde yer almaktadır. Bir bean, Çekirdekten Yansıma API'sini kullanarak sınıfları, arayüzleri, metodları ve parametrelerini elde eden İçgözlemci sınıfı aracılığı ile incelenebilir. Toplanan veriler, bir BeanInfo nesnesi içinde geri döndürülür. BeanInfo ise, nesneyi diğer bileşenlere bağlamak ve özelliklerini değiştirmek için bir uygulama geliştirme aracı tarafından kullanılabilir.

Kurucuları ve metodları dinamik olarak elde etme ve çağırma imkanı çok güçlü bir özelliktir. Bu tür programlama yeteneklerinin metod yapıları ve programlama özellikleri ile birleşmesiyle, bean'leri programlara kod yazmadan bağlamak mümkün olacaktır.

2.3.3 BeanInfo Arayüzü ve Tasarım Yapıları

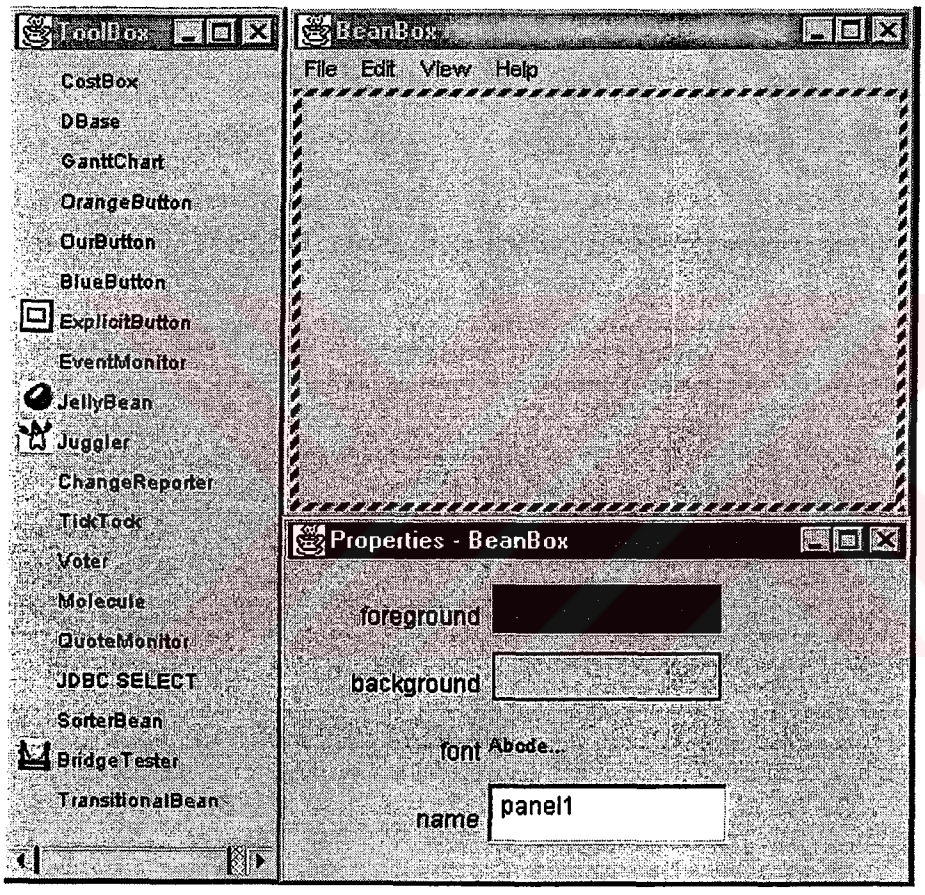
Tasarım yapıları, bir programın parçalarını tutarlı bir şekilde temsil etmek için kullanılan basit yöntemlerdir. Düzenler, Çekirdekten Yansıma API'si kullanılarak farkedilecek şekilde tasarlanmıştır. İçgözlemci sınıfı, Çekirdekten Yansıma API'sini kullanarak bir Bean'de yer alan metodları elde eder ve bu bilgiyi bir BeanInfo nesnesine yerleştirir. Bir BeanInfo nesnesi, BeanInfo arayüzünün gerçeklenmesidir. BeanInfo arayüzü, bir Bean'i tanımlayan Bean açıklayıcılarını, özelliklerini, metodlarını ve olayları elde etmek için kullanılan metodları tanımlar. BeanInfo sınıfı, bir Bean'i düzenlemek için bir uygulama geliştirme aracı tarafından kullanılabilen bir Bean'in standart görünümü olarak kullanılabilir ve metodlarını, özelliklerini ve olaylarını tanıtır. BeanInfo, geliştirici tarafından oluşturulabilir veya Inspector sınıfı kullanılarak da elde edilebilir.

2.4 Beanbox'ın Tanımı

BDK (Beans Developer Kit) ile birlikte gelen bir uygulama olan BeanBox, JavaBean uyumlu uygulama tasarım araçlarının nasıl kurulacağını gösteren bir örnektir. BeanBox içinde, JavaBean'ler canlı nesneler olarak düzenlenebilirler.

BeanBox, oldukça güçlü bir Bean GUI düzenleyici olarak kabul edilebilir.

Beanbox'ın çalıştırıldığındaki ilk görüntüsü aşağıdaki gibidir.



Şekil 2.1 Beanbox'ın görünümü

2.5 JavaBean Türleri

Dört tür Bean mümkündür :

- Görüntü veren bileşen

- Taşıyıcı

- Görünmeyen bileşen

- Applet

Bu türler, aslında Bean API tarafından doğrudan desteklenmemektedirler. Bir görüntü veren bileşen bean'i, bileşen yapan tek şey, java.awt.Component sınıfından miras almasıdır.

2.5.1 Görüntü Veren JavaBean'ler

Component ve Container sınıflarının çocuk sınıfları olan bean'lere, aynı zamanda görünebilir bean'ler de denir, çünkü GUI uygulamaları geliştirmek için kullanılırlar. Görünebilir bean'ler, öncelikle Frame, Applet ve Panel nesnelerinin dahil olduğu Container nesnelerine eklenirler.

2.5.2 Görünmeyen JavaBean'ler

Görünmez JavaBean'lerin görünmez olmalarının sebebi, Component sınıfına bağlı bir sınıftan miras almıyor olmamarıdır. Ama kendileri pencereler veya diyaloglar gibi çerçeve temelli bileşenler oluşturabilirler. Özellikleri olan herhangi bir sınıf, görünmez bir bean olabilir.

2.5.3 Applet JavaBean'ler

Applet JavaBean'ler, yeni bir tip JavaBean bileşenidir. Bir applet, Container sınıfının bir uzantısı olduğundan bir taşıyıcı veya bir bileşen şeklinde görülebilir.

Applet JavaBeanler, diğer Bean'lerden farklıdırlar, çünkü içinde çalışacakları bir bağlama ihtiyaç duyarlar. Gerekli bağlamı yaratmak için, BeanBox veya başka bir uygulama geliştirme aracı AppletContext arayüzünden miras alan bir sınıf oluşturur. Applet, burada mevcut metodları kullanarak, uygulama içindeki diğer Applet temelli JavaBean'lerin kaynaklarına ve ilişkilerine erişebilir. Ayrıca AppletStub arayüzünü uygulayan bir taşıyıcı sınıf da yaratılır. Bu sınıf sayesinde, Applet nesnesi ve Applet parametreleri elde edilir.

2.6 JavaBean'lerin Dağıtımı

JavaSoft'un, JavaBean'lerle ilgili bütünlüğü, sınıf ve destek dosyalarının paketlenmesini de içerir. Paketleme ve tekrar açma metodları da Java yazılımı tarafından gerçeklenmektedir. Java-tabanlı araçlar ve kütüphaneler sayesinde, bu çözüm, Java platformları arasında tamamen taşınabilirdir. Paketleme mekanizması, iki bölümden oluşur, JAR dosyaları ve Bildirim (Manifest) dosyaları.

2.6.1 Jar Dosyaları

Yazılımların sıkıştırılmış arşiv dosyalarında dağıtılması bir gelenek halini almıştır. Bu, JAR dosyaları olarak adlandırılan bir dosya tipi ve sıkıştırma sistemi kullanan JavaBean'ler için de doğrudur. Bean'ler için kullanılan JAR dosyaları, ayrıca Java applet'lerini ve uygulamalarını sıkıştırmak için de kullanılmaktadır. "jar" (95/NT veya DOS için "jar.exe") adındaki araç, dosyaları arşivlemek veya bir JAR dosyasını açmak amacıyla kullanılmaktadır.

JAR dosyalarının yapısı, ZIP dosyalarının standart yapısını temel alır. JAR aracı, arşivlenen dosyaların dizin ağacını otomatik olarak saklar. Dizin ağacı, Java kodu için oldukça önemlidir, çünkü sınıf dosyalarının paket isimlerini belirlemek için kullanılırlar.

Bir arşivleme sistemi eklemenin birincil nedeni, platform-bağımsız ve Java Çekirdek API'de yer alan java.util.zip paketi tarafından desteklenen bir standart format sağlamaktır. Java Çekirdek API, tüm Java kurulumları içinde yer aldığından, JAR dosyalarını okumak veya yaratmak için ek bir yazılıma ihtiyaç yoktur.

2.6.2 Bildirim Dosyaları

Bir bildirim dosyası, bir JAR dosyasında yer alan tüm dosyaların basit bir açıklamasıdır. Bildirim dosyası, her dosyanın adını, checksum'ı ve sınıfları Bean olarak tanımlayabilmek için gerekli bilgileri içerir. Ayrıca kullanıcı, JAR dosyasındaki verileri düzenlemeye yardım edecek ek bilgiler sağlayabilir.

Bildirim dosyasının en önemli noktası, bir Bean'in ana sınıfı olan sınıf dosyasının "Java-Bean:True" satırıyla işaretlenmesi gerektiğidir. Bu satır eklendiğinde, IDE

araçları ve BeanBox uygulaması gibi diğer Bean-haberdar uygulamalar, doğru Bean sınıfını ayırt edebilirler.

2.7 Bean'lerin Kullanım Türleri

JavaBean uygulamaları, farklı kullanımlar ve düzenlerde olabilir.

2.7.1 Bean'lerde Uyumluluk

Bean'ler çok yönlüdürler. Bir bileşenin çok farklı özelliklere sahip, farklı tiplerini oluşturmak mümkündür. Bunun yanında, farklı görüntülerde veya farklı işlem modlarında olabilecek şekilde düzenlenebilecek tek bir Bean de oluşturulabilir. Karmaşık Bean'leri oluşturmak için bir araya gelen Bean grupları mevcut olabilir. Amaç, bir Bean'i, istenen işler kümesi halinde konfigüre edilebilecek şekilde tasarlamaktır.

2.7.2 Bean'ler ve Yazı Dilleri

Bean API'nin yapısı, Bean'lerin yazılı dillerde ve uygulamalarda kullanılmasını anlamlı kılar. Netscape, JavaScript dilini geliştirerek, JavaBean'lerin kullanımına izin vermeyi planlamaktadır. Bu sayede, yorumlanan dilin basitliğinden vazgeçmeden geliştirilmesi sağlanmış olur.

2.7.3 Bean Köprüleri : Bean'ler ve Programlama Dilleri Arasında Arayüzler

C++ veya Visual Basic gibi diğer temel diller de JavaBean'leri kullanabilirler. İlgili köprüleri kullanarak, bu diller ve JVM'de koşan Bean'ler arasındaki arayüz sağlanabilir. Bu işlem ters yönde de geçerlidir. Yani Java uygulamaları, diğer bileşen teknolojilerini kullanabilirler. Mümkün oldukça, bundan kaçınılmalıdır. Java olmayan koda yer verildiğinde, güvenlik açısından problemlerle karşılaşılabilir. Taşınabilirlik, hata kontrolü ve güvenlik için %100 Java'dan vazgeçmemek en iyisidir.

3. JAVABEAN OLUŞTURULMASINDA TEMEL BİLGİLER

3.1 JavaBean Tasarım Yapıları

Bir bean yazarken dikkat edilmesi gereken en önemli nokta, bazı yazım kurallarına uymaktır. "Tasarım Yapıları" olarak adlandırılan bu kurallar, aslında programlama yapılarıdır. JavaBeans Introspector (İçgözlem) sınıfı, Java Çekirdekten Yansıma API'sini kullanarak, bean'in bölümlerini uygun şekilde sınıflandırır.

Tasarım yapıları, dört farklı sınıfa ayrılırlar: özellik erişimi, olay kaynakları, olay dinleyiciler ve JavaBean'ler. Özellik erişimi yapıları, Bean özelliklerini belirlemek veya almak (set/get) için harici erişimi sağlarlar. Olay kaynak yapıları, bir Bean'in ürettiği olayları, olay dinleyici yapıları ise, Bean'in işleyebileceği olayları tanımlar. JavaBean yapıları ise, bir Bean'in diğer sınıflara erişimini genişletmek için, JavaBean-tanıyan yazılımlar tarafından kullanılan sınıflardır. Tasarım yapıları, İçgözlem konusu açıklanırken, daha detaylı bir şekilde incelenecektir.

3.2 Özellikler

Özellikler, bir JavaBean'in görünüşünü ve davranışlarını etkileyen, sonlu nitelikleridir. Bir başka deyişle özellikler, bir bean'in iç durumunu gösterirler. Örneğin, bir buton, üzerinde görüntülenen metni temsil eden, "etiket" isminde bir özelliğe sahip olabilir.

Özellikler, birkaç şekilde karşımıza çıkarlar:

1. Diğer bileşenler tarafından, alıcı ("getter") ve belirleyici ("setter") metodlarını çağırarak erişilebilirler.

2. Bir bileşeni yerel uyarlama işleminin bir parçası olarak, bu bileşenin özellikleri, kullanıcının düzenleyebilmesi için bir özellik sayfasında sunulabilirler.

3. Bir bean'in özellikleri kalıcı olmalıdırlar. Böylece son durumları, bean'in kalıcı durumunun bir parçası olarak saklanacaktır.

Özellikler, "int" gibi yerleşik Java tipleri ve "java.awt.Color" gibi sınıf veya arayüz tipleri dahil olmak üzere farklı tiplere sahip olabilirler.

3.2.1 Erişim Metodları

Özelliklere erişmek için, içinde yer aldıkları nesnenin metod çağrıları kullanılır. Okunabilen özellikler için, özelliğin değerini alabilmek üzere bir "alıcı" (getter) metod bulunur. Değiştirilebilen özellikler için ise, özelliğin değerini güncellemeye izin veren bir "belirleyici" (setter) metod bulunur.

Özelliklerin sadece basit veri alanları olması gerekli değildir, hesaplanan değerler de olabilirler. Bir özelliğin değerinde yapılan bir güncellemenin, programlama açısından birçok etkisi olabilir. Bu etkiler, özellik türleri ele alındığında, daha detaylı bir şekilde açıklanacaktır.

Basit özellikler için, erişim tipi yapıları şöyledir :

```
void setFoo(ÖzellikTipi value);      // basit belirleyici (setter)

ÖzellikTipi getFoo();                // basit al c  (getter)
```

Burada verilen erişim metod isimleri sadece birer örnektir ve bu metodlar, daha sonra İçgözlem konusunda detaylı bir şekilde incelenen tasarım yapılarına uygun olmak şartı ile çok farklı isimler alabilirler.

3.2.2 İndeksli Özellikler

Bir bean, basit, tek değerli özelliklerden başka, indeksli özelliklere de sahip olabilir. Bu tür özellikler, geleneksel Java programlamada yer alan dizilere çok benzerler. İndeksli özellikler, Java int yerleşik tipinden olan tamsayı bir indeks ile ulaşılabilen aynı tipten birçok elemandan oluşurlar.

İndeksli özelliklerle ilgili önemli bir nokta da, bu özelliklere ait erişim metodlarının, hem tüm diziye ulaşma, hem de istendiğinde dizinin elemanlarına indeks aracılığı ile tek tek ulaşma imkanı vermeleridir.

İndeksli özellikler için, erişim tipi yapıları aşağıdaki gibidir :

```
void setter(int indeks, ÖzellikTipi value);      // indeksli setter

ÖzellikTipi getter(int indeks);                  // indeksli getter

void setter(ÖzellikTipi values[]);               // dizi setter

ÖzellikTipi[] getter();                          // dizi getter
```

3.2.3 Bağlı Özellikler

Bir bean özelliği değiştiğinde, bean'in taşıyıcı nesnesi veya başka bir bean, değişiklikten haberdar olmak isteyebilir.

Bir nesne, özelliklerinin bir kısmı veya hepsi için böyle bir değişiklik haber verme mekanizmasını sağlayabilir. Bu tür özellikler, `bağlı (bound) özellikler` olarak bilinirler.

Basit bağlı özelliklere yapılan güncellemeleri bildirmek için `PropertyChangeListener` olay dinleyici arayüzü kullanılır. Eğer bir bean, bağlı özellikleri destekliyorsa, özellik değişimini dinlemek isteyen sınıfların kendilerini kayıt ettirebilecekleri bir çift kayıt metodunu da içermelidir :

```
public void addPropertyChangeListener(PropertyChangeListener x);

public void removePropertyChangeListener(PropertyChangeListener x);
```

Bağlı bir özellik üzerinde bir değişim olduğunda, bean kayıtlı dinleyicilerin `PropertyChangeListener.propertyChange` metodunu, özelliğin yerel adını, eski ve yeni değerlerini içine alan bir `PropertyChangeEvent` nesnesi ile çağırır.

Olayın kaynağı olan, bağlı özelliği içeren bean, kendi içinde gerekli güncellemeleri yaptıktan sonra olayı ateşlemelidir.

Programlamada rahatlık sağlaması için, `PropertyChangeSupport` yardımcı sınıfı tanımlanmıştır. Bu sınıf, `PropertyChangeListener` sınıflarını izlemek ve `PropertyChangeEvent` olaylarını ateşlemek için kullanılır.

3.2.4 Zorlanan Özellikler

Bir özellik değiştiğinde, bir başka bean, değişimi doğrulamak ve uygun bulmadığı zaman geri çevirmek isteyebilir. Bu tür bir kontrolden geçen özelliklere `zorlanan (constrained) özellikler` denir.

Bir bean'de yer alan zorlanan özelliğe ait belirleyici metodların `PropertyVetoException` özel durumunu desteklemeleri gereklidir. Bu, zorlanan özellik kullanıcılarına, güncelleme girişimlerinin reddedilebileceğini haber verir.

Zorlanan özelliklere yapılan güncellemeleri raporlamak için `VetoableChangeListener` olay dinleyici arayüzü kullanılır. Eğer bir bean, zorlanan özellikleri destekliyorsa, veto edilebilir değişimleri dinlemek isteyen sınıfların kendilerini kayıt ettirebilecekleri bir çift kayıt metodunu da içermelidir :

```
public void addVetoableChangeListener(VetoableChangeListener x);

public void removeVetoableChangeListener(VetoableChangeListener x);
```

Zorlanan bir özellik üzerinde bir değişim olduğunda, bean kayıtlı dinleyicilerin `VetoableChangeListener.vetoableChange` metodunu özelliğin yerel adını, eski ve yeni değerlerini içine alan bir `PropertyChangeEvent` nesnesi ile çağırır. Eğer alıcı, istenen düzenlemenin yerine getirilmesini onaylamazsa, bir `PropertyVetoException` özel durumunu oluşturur. Bu özel durumu yakalamak, özelliği eski değerine geri döndürmek, ve geri almayı haber vermek üzere yeni bir `VetoableChangeListener.vetoableChange` olayı yayınlamak kaynak bean'in sorumluluğudur.

Olayın kaynağı olan, zorlanan özelliği içeren bean, kendi içinde gerekli güncellemeleri yapmadan önce olayı ateşlemelidir.

Programlamada rahatlık sağlaması için, `VetoableChangeSupport` yardımcı sınıfı tanımlanmıştır. Bu sınıf, dinleyici listesine `VetoableChangeListener` nesnelerini eklemek ve kaldırmak için metodlar ve bir özellik değişimi olduğunda, bu listedeki her dinleyiciye özellik değişim olaylarını ateşlemek için bir metod gerçekleştirir. Aynı zamanda, `PropertyVetoException` özel durumlarını da yakalayıp, gerekli geri alma olaylarını da gönderir.

Genellikle, zorlanan özelliklerin aynı zamanda bağlı olmaları da önerilir. Eğer bir bean, hem bağlı hem de zorlanan bir özelliğe sahipse, özelliği güncellemeden önce bir `VetoableChangeListener.vetoableChange` olayı, ve özelliği güncelledikten sonra bir `PropertyChangeListener.propertyChange` olayı ateşlemelidir.

Eğer bir bean, bir özelliğin durumu ile ilgili, hem teklif edilen güncellemeleri reddedebilmek hem de doğru bir şekilde takip edebilmek istiyorsa, "iki evreli" bir yaklaşımı tercih etmeli ve hem bir `VetoableChangeListener`, hem de bir `PropertyChangeListener` olarak kayıt olmalıdır. İlk evrede, teklif edilen özellik değerinin kabul edilip edilemeyeceğini kontrol etmeli ve kabul edilemeyecek değerleri veto etmelidir. İkinci evrede ise, değişimin etkin olduğunu kabul ederek, yeni değeri kullanabilir.

Eğer bir özelliğe ait belirleyici (setter) metod, özelliğin mevcut değerine eşit bir parametre ile çağırılırsa, bean'in `VetoableChangeListener.vetoableChange` veya `PropertyChangeListener.propertyChange` olaylarını ateşlememesi tavsiye edilir. Bu davranışın sonucu olarak hem daha fazla verim elde edilir, hem de belirleyici metodların çapraz bağlanmış olması durumunda döngü tehlikesi azaltılmış olur. `PropertyChangeSupport` ve `VetoableChangeSupport` yardımcı sınıflarını kullanan kodlarda, bu kontrol otomatik olarak yapılmaktadır, çünkü bu sınıflar, "eski" ve "yeni" özellik değerlerinin eşit olup olmadığını kontrol ederler ve eşitlik olduğu durumlarda bir olay ateşlemezler.

3.3 JavaBean'ler ve Olaylar

Olaylar, JavaBean yapısının en temel özelliklerinden biridir. Olaylar, bazı bileşenlerin olay bildirimleri için kaynak görevi görürken, bazılarının ise, bu olayları yakalayıp işlemesine izin vererek, bileşenlerin uygulama geliştirme araçlarında birbirlerine bağlanabilmeleri için uygun bir mekanizma sağlarlar.

Tüm olayların temeli, `java.util.EventObject` sınıfıdır ve tüm olaylar tarafından kullanılacak işlevsellikleri içerir. `EventObject` aynı zamanda, olay kuyrukları ve olay adaptörleri gibi olay işleyicileri için kullanılan genel bir sınıf görevi de görür.

3.3.1 Olay Temelleri

Olaylarla ilgili anahatlar aşağıdaki gibi listelenebilir :

- Olay uyarıları, hedef dinleyici nesneleri üzerinde yapılan Java metod çağrıları ile yayılırlar.

- Her ayrı olay uyarı tipi, ayrı bir Java metodu olarak tanımlanır. Bu metodlar, `java.util.EventListener` 'dan miras alan `EventListener` arayüzleri halinde gruplanırlar.

- Olay dinleyici sınıflar, bazı `EventListener` arayüzlerini gerçekleyerek, belirli olaylara olan ilgilerini belirtirler.

- Bir olay uyarısı ile ilişkilendirilen durum, normalde, `java.util.EventObject` 'den miras alan ve olay metoduna geçirilen tek parametre olan bir olay durum nesnesi içine yerleştirilirler.

- Olay kaynakları, belirli tasarım yapılarına (daha ileride detaylı bir şekilde ele alınacaktır) uyan kayıt metodları tanımlayarak, kendilerini bazı olaylara kaynak olarak belirtirler ve belirli `EventListener` arayüz örneklerine referanslar kabul ederler.

- Dinleyicilerin doğrudan bir arayüzü gerçekleyemedikleri veya bazı ek davranışların gerekli olduğu durumlarda, bir kaynak ve bir veya birden fazla dinleyici arasındaki ilişkiyi kurmak için, özel bir adaptör sınıfı örneği kullanılabilir.

3.3.2 Temel Olay Türleri

Java'da iki tür olay tanımlıdır : alt-düzey olaylar ve anlamsal olaylar.

Alt-düzey olaylar, doğrudan fare, tuştakımı, bileşen, taşıyıcı ve pencere bilgileri ile ilişkilidir. Yani, genellikle donanım ve bileşen ve pencerelerin varsayılan davranışları ile ilişkililerdir. Anlamsal olaylar ise, bir kullanıcı arayüz bileşen modelinin anlamını içine almak için kullanılırlar.

Örneğin, bir butona kliklediğinizde veya bir listede yer alan bir elemanın üzerine çift-kliklediğinizde, bir işlemin yürütülmesini istersiniz. Ama bir kaydırma çubuğu üzerine kliklediğinizde bir değeri ayarlamak istersiniz. Buradan, fareye klikleme ile oluşturulan alt-düzey olay, farklı bileşenler için aynı görevi yerine getirmemektedir. Bu yüzden, anlamsal olaylara ihtiyaç duyulmaktadır.

3.3.3 Olay Durum Nesneleri

Belirli bir olay bildirimi ile ilgili bilgiler, normalde, `java.util.EventObject` 'in bir alt sınıfı olan bir olay durum nesnesinin içinde tutulurlar. Kolaylık sağlaması için, bu olay durum sınıflarına "Event" ile biten isimler verilir. Olay durum nesneleriyle ilgili, alanlarına erişmek gibi, işlemler yapılmak istendiğinde, doğrudan erişim yerine, erişim metodlarının kullanımı tavsiye edilmektedir.

3.3.4 Olay Dinleyici Arayüzleri

Olay işleme metodları, `java.util.EventListener`'dan miras alan olay dinleyicilerinde tanımlanırlar. Kolaylık sağlaması için, bu arayüzlere "Listener" ile biten isimler verilir.

Verilen bir olay dinleyici arayüzünde tanımlı olaylardan herhangi birini kabul edip işlemek isteyen bir sınıf, bu arayüzü gerçeklemelidir. Olay dinleyici arayüzlerinde tanımlı olay işleme metodları, standart bir tasarım yapısına uymalıdırlar. Bu sayede, olay sisteminin yardım ve dökümantasyonu ile ilgili kolaylık sağlanmış olur. Ayrıca, arayüzlerin üçüncü parti yazılımlar tarafından farkedilmelerine ve olay adaptörlerinin otomatik oluşturulmasına izin verir.

3.3.5 Olay Dinleyici Kayıtları

Olay kaynak sınıfları, olası olay dinleyicilerin, uygun olay kaynağı örneğine kaydolmaları ve kaynaktan dinleyiciye bir olay zinciri oluşturabilmelerini sağlamak üzere, olay dinleyicilerin kayıtlarını eklemek ve silmek için metodlar sağlamalıdırlar.

Kayıt metodları, belirli bir tasarım yapı kümesine uygun olmalıdırlar. Bu sayede, dökümantasyon kolaylaşır ve JavaBean içgözlem API 'leri ve uygulama geliştirme araçları için programlanabilir içgözleme izin verilir.

Olay dinleyici kayıdı için standart tasarım yapısı şöyledir :

```
public void add<DinleyiciTipi>(<DinleyiciTipi> dinleyici);

public void remove<DinleyiciTipi>(<DinleyiciTipi> dinleyici);
```

`add<DinleyiciTipi>` metodunu çağırmak, verilen dinleyiciyi, `DinleyiciTipi` ile ilişkili olaylara kayıtlı, olay dinleyicileri kümesine ekler. Benzer şekilde, `remove<DinleyiciTipi>` metodunu çağırmak, verilen dinleyiciyi, `DinleyiciTipi` ile ilişkili olaylara kayıtlı, olay dinleyicileri kümesinden siler.

3.3.6 Olay Dağıtımı

Normalde, olay dağıtımı çokludur. Yani, bir olay dinleyicisi üzerinde birden fazla dinleyici tanımlanabilir. Ama bazı zorunlu hallerde, dinleyici sayısının birden fazla olması sorun yaratabilir. Böyle durumlarda, sadece bir dinleyicinin kendini kayıt ettirmesine izin verilir.

Her çoklu olay kaynağı, ateşlediği her farklı türdeki olay için, bir olay dinleyiciler kümesi tutmalıdır. Bir olay meydana geldiğinde, olay kaynağı, seçilebilir her hedef dinleyiciyi çağırır. Varsayılan olarak, mevcut tüm kayıtlı dinleyiciler, bildirim için 'seçilebilir' olarak kabul edilirler. Ancak herhangi bir kaynak, kaynağın, dinleyicilerin, sistemin bir bölümünün mevcut durumunu veya olayın kendi durumunu temel alan bir seçim kriterine bağlı bir gerçekleme kullanarak, seçilebilir dinleyiciler kümesini, kayıtlı dinleyicilerin bir altkümesiyle sınırlandırabilir.

Eğer olay dinleyicilerin herhangi biri bir özel durum oluşturursa, olayı geri kalan dinleyicilere dağıtıp dağıtmamak, olay kaynağının gerçeklemesiyle ilgilidir.

Bazı olay kaynakları sadece tekli dinleyici kayıdı ve olay dağıtımını kabul edebilirler. Bu durumda, olay kaynağı, ateşlediği her farklı tekli olay için, tek bir olay dinleyicisi tutmalıdır. Tekli bir olay meydana geldiğinde, olay kaynağı, bu tek hedef dinleyiciyi çağırmalıdır.

3.3.7 Olay Adaptörleri

Olay adaptörleri, Java olay modelinin çok önemli bir parçasını oluştururlar.

Bazı uygulamalar veya uygulama geliştirme araçları, olay dağıtımı üzerinde ek bir idare sağlamak için, olay kaynakları ve olay dinleyicileri arasında bir olay adaptörleri kümesi kullanabilirler.

Olay dağıtımı sırasında ek bir davranışa gerek duyulduğunda, arada bir "olay adaptör" sınıfı tanımlanabilir ve bir olay kaynağı ile gerçek olay dinleyici arasına yerleştirilebilir.

Bir olay adaptörünün başlıca görevi, olay kaynağının beklediği belirli olay dinleyici arayüzüne uygun olmak ve arayüz üzerinde, gelen olay bildirimlerini gerçek dinleyicilerden ayırmaktır.

Bu tür adaptörlere ait bazı örnek kullanımlar:

- Kaynaklar ve dinleyiciler arasında bir olay kuyruğu mekanizması gerçekleştirmek.

- Bir filtre görevi görmek.

- Birden fazla olay kaynağını tek bir olay dinleyicisine bağlamak.

- Kaynaklar ve dinleyiciler arasında bir "bağlantı yöneticisi" görevi görmek.

3.4 JavaBean İçgözlem

Bir geliştirme ortamında veya koşma anında, bir Java bean'in hangi özellikleri, olayları ve metodları desteklediği bilgisini elde edebilmeliyiz. Bu sürece, `içgözlem (introspection)` adı verilmiştir.

Bir geliştiricinin tamamen Java destekli bir ortamda çalışabilmesi ve bu yüzden de bir Java bean'in davranışını belirlemek için ayrı bir tanımlama diline ihtiyaç duymasının engellenmesi istenir. Bu davranışın, Java'da tanımlanabilmesi sağlanmıştır.

JavaBean API'si tarafından sağlanan içgözlem hizmetleri iki bölümde incelenebilir: alt-düzey ve üst-düzey hizmetler.

Alt-düzey API hizmetleri, bazı ileri geliştirme özellikleri sağlayabilmek için bean'in iç yapısı üzerinde yoğunlaşan uygulama geliştirme araçları tarafından kullanılırlar. Ancak, bu erişim düzeyi, uygulama geliştiriciler için uygun değildir, çünkü uygulama düzeyinde erişime açık olmayan bean'in iç yapısıyla ilgili bazı bilgiler sunulmaktadır.

Üst-düzey hizmetler, bean'in iç yapısıyla ilgili bilgilerin, public (genel) özellikler, metodlar ve olaylardan oluşan, kısıtlı bir bölümüne erişim sağlarlar. Ancak yine bu servisler de büyük oranda alt-düzey hizmetlere dayanırlar.

Bazı bileşen modellerinde, bileşeni geliştiren kişinin, dışarıdan erişimin mümkün olmasını istediği her nokta ile ilgili, gerekli tanımları tek tek yapması gerekmektedir. Bu süreç, çok zorlu olabilir ve hatta bir bileşen tanımlama dilini öğrenmeyi bile gerektirebilir. Bu noktada, JavaBean'leri çok farklı bir yaklaşıma sahiptirler. Bean'e ait bilgileri, doğrudan kendi sınıf yapısından otomatik olarak elde etmek mümkündür. Bu, bean geliştirici yerine, JavaBean İçgözlem sınıfına bazı sorumluluklar yüklenmesi demektir.

Yalnız JavaBean'lerde içgözleme tek yaklaşım yolu, otomatik içgözlem değildir. Eğer bileşen geliştirici, bean bilgileri ile ilgili daha özel tanımlamalar yapmak isterse, dışarıya açık olmasını istediği bilgileri kendisinin açıkça tanımlama seçeneği de vardır. Bu yaklaşım da, diğer bileşen modellerinde gereken yapıya göre oldukça kolaydır.

Yukarıda bahsedilen bir bean hakkında bilgileri otomatik elde etme yöntemi, çekirdek Java API'de yer alan yansıma (reflection) kolaylıklarını kullanmaktadır. Bu yansıma kolaylıkları, bir bean'e ait alt-düzey yapıyı incelemek ve bilgi döndürmekle yükümlüdürler. JavaBean API'sindeki içgözlem araçları ise, bu bilgiyi alır ve bir bean tarafından desteklenen genel (public) özellikler, metodlar ve olayların niteliklerini elde edebilmek için, tasarım yapılarını buna uygularlar. Bu yaklaşımda, geliştiriciye düşen görev, bean'ine ait kodu yazarken aşağıda daha detaylı ele alınan tasarım yapılarına uymaktır.

İkinci yaklaşım ise, geliştiriciye biraz daha fazla görev yüklemektedir. Bu yaklaşımı kullanıp kullanmamak geliştiricinin seçimine bağlıdır ve otomatik yaklaşımla birlikte de kullanılabilir. Bu yaklaşımda, geliştiricinin bir BeanInfo sınıfı oluşturması gereklidir. JavaBean API'si bu işlemde kullanılmak üzere destek sınıflar sağlamaktadır.

3.4.1 Özellikler İçin Tasarım Yapıları

3.4.1.1 Basit Özellikler

Varsayılan olarak, tasarım yapıları kullanılarak aşağıda verilen formda metodlar aranır ve özellikler elde edilmeye çalışılır.

```
public <ÖzellikTipi> get<ÖzellikAd >();

public void set<ÖzellikAd >(<ÖzellikTipi> a);
```

Eğer birbirine uyan "get<ÖzellikAdı>" ve "set<ÖzellikAdı>" metod çifti varsa, "<özellikAdı>" adında bir oku-yaz özelliği tanımlı demektir.

"get<ÖzellikAdı>" metodu, özelliğin değerini elde etmek ve "set<ÖzellikAdı>" metodu ise, özelliğin değerini ayarlamak için kullanılırlar.

Eğer bu metodlardan sadece biri mevcutsa, "<özellikAdı>" isimli bir salt-oku veya salt-yaz özellik tanımlanmış demektir.

Bir örnek verirsek, `long` tipinden `numBullets` özelliği için içgözlem tasarım yapılarına uygun oku-yaz metodları aşağıdaki gibidir :

```
public long getNumBullets();

public void setNumBullets(long nb);
```

3.4.1.2 Boolean Özellikler

Boolean özellikler için, basit özelliklerden farklı olarak, yapıya uyacak yeni bir alıcı metod tanımlanır:

```
public boolean is<ÖzellikAd >();
```

Bu "is<ÖzellikAdı>" metodu, "get<ÖzellikAdı>" metodunun yerine tanımlanabilir veya "get<ÖzellikAdı>" metoduna ek olarak sağlanabilir. Herhangi bir durumda, eğer boolean bir değişken için "is<ÖzellikAdı>" metodu mevcutsa, özelliğin değerini okumak için bu metod kullanılır.

Örnek bir boolean özelliğin erişim metodları aşağıdaki gibidir:

```
public boolean isEnglish();

public void setEnglish(boolean e);
```

3.4.1.3 İndeksli Özellikler

Eğer dizi "<ÖzellikElemanı>[]" tipinde bir özellik mevcutsa, aşağıda verilen formda metodların da bulunması gereklidir :

```
public <ÖzellikEleman > get<ÖzellikAd >(int a);

public void set<ÖzellikAd >(int a, <ÖzellikEleman > b);
```

Eğer bu yapıların biri tanımlıysa, "<ÖzellikAdı>"nın indeksli bir özellik olduğunu ve bu metodların bir indeksli değeri okumak ve/veya yazmak için kullanılabileceğini anlarız.

`Color` nesnelerinden meydana gelen bir dizi olan `palet` adındaki, indeksli özelliğin erişim metodlarını oluşturursak :

```
public Color getPalet(int i);

public void setPalet(int i, Color c);

public Color[] getPalet();

public void setPalet(Color[] c);
```

3.4.2 Olaylar İçin Tasarım Yapıları

Bir bean'in iç durumu değiştiğinde, genellikle dışardan bir ilgiliyi bu değişimden haberdar etmek gerekir. Bu sebeple, bean'ler gerektiğinde, olay bildirimlerini ilgililere iletme yeteneğine sahiptirler.

Bean'ler, olay bildirimlerini almak istediklerini söyleyerek, kendilerini kayıt ettiren olay dinleyicilerine bildirimleri gönderirler. Dinleyicilerin kendilerini bean'e kayıt ettirmeleri gerektiğinden, bean'lerin kayıt işlemleri için gerekli metodları sağlamaları önemlidir. Olay kayıt metodları çiftler halindedirler, dinleyicilerin eklenmesine izin veren bir metod ve dinleyicilerin silinmesi için bir metod. Bu metodlar, bir bean ve ilgili başkası arasındaki olay kayıdını tanımlamak için yeterlidirler ve bir bean'in gönderebileceği olayları belirlemek için JavaBean içgözlem mekanizması tarafından kullanılırlar.

Otomatik içgözlem hizmetlerinin doğru bir şekilde çalışabilmesi için, olay kayıt metodlarının aşağıda tanımlanan tasarım yapılarına uymaları gereklidir :

```
public void add<OlayDinleyiciTipi>(<OlayDinleyiciTipi> x);

public void remove<OlayDinleyiciTipi>(<OlayDinleyiciTipi> x);
```

Bu metodların ikisi de, "java.util.EventListener" arayüzünden türetilen aynı `<OlayDinleyiciTipi>` nden değişken alırlar. `<OlayDinleyiciTipi>` tip adı, "Listener" sözcüğü ile bitmelidir.

Yukarıda verilen olay kayıdı tasarım yapıları, çoklu olayları destekleyen bean'ler için tasarlanmışlardır. Bu durumda, birden fazla kayıtlı olay dinleyicisi vardır.

Ancak JavaBean'leri, bir anda olay bildirimlerini alacak sadece bir dinleyiciye izin veren bean'ler demek olan tekli olay kaynaklarını da desteklerler. Yani tekli bean'ler, belirli bir zamanda sadece bir olay dinleyicisinin kayıtlı olmasına imkan verirler. Eğer birden fazla kullanıcı, bir tekli bean'e kayıt olmaya çalışırsa, `add<OlayDinleyiciTipi>` metodu, `java.util.TooManyListenersException` özel durumunu oluşturur. Bu tür bean'lerin içermeleri gereken kayıt metodları aşağıdaki gibi olmalıdır :

```
public void add<OlayDinleyiciTipi>(<OlayDinleyiciTipi> x) throws
                        java.util.TooManyListeners;

public void remove<OlayDinleyiciTipi>(<OlayDinleyiciTipi> x);
```

3.4.3 Metod Tasarım Yapıları

Bu bölüme, erişim metodu olmayan genel metodlar dahildir. Bu metodların bir özelliğin değerini almak veya ayarlamak gibi önceden tanımlı bir amaçları yoktur, tanımları tamamen kullanıcıya bağlıdır.Bu sebeple, bu tür metodlar için özel bir tasarım yapısı tanımlanmamıştır. Tüm genel (public) metodların dışarıdan erişime açıldığı kabul edilir. Bu genel metodlar ve genel erişim metodları arasındaki tek fark, erişim metodlarının, JavaBean içgözlem mekanizmasına bir bean özelliğini bildirmesidir.

3.4.4 Bir Beanınfo Sınıfı Kullanarak Bean Bilgilerini Açıkça Sunmak

Bir bean, desteklediği özellikleri, olayları ve metodları, BeanInfo arayüzünü gerçekleyen bir sınıf sunarak, açıkça da belirtebilir.

BeanInfo arayüzü, hedef bir bean'in olayları, özellikleri, metodlarını öğrenmek ve kendi hakkında bazı genel bilgiler elde etmek için metodlar sağlar.

Bir bean'in bilgi sınıfının adı, bean'in sınıf adının sonuna `BeanInfo` eklenerek oluşturulur.

Bean bilgilerini açıkça sunmak isteyen geliştiriciler, istedikleri takdirde, `BeanInfo` arayüzü aracılığı ile kısmi bir bilgi sunma şansına da sahiptirler. Diğer bir deyişle, örneğin, `BeanInfo` arayüzünü kullanarak bir bean'in metodları hakkında bilgi sağlarken, olayları hakkında bilgi sunmamak mümkündür. Bu durumda, JavaBean içgözlem mekanizması, bean'in metodlarını incelerken, açıkça sunulan bilgiyi kullanacak, bean'in olayları ve özellikleri söz konusu olduğunda ise, bu bilgiyi bulamadığından otomatik içgözlemi kullanacaktır.

3.4.5 Bir Bean'in Analizi

JavaBean'lerde hem bir bean'in dışarıya sunulan özellik, metod ve olaylarının açıkça sunulmasına, hem de tasarım yapıları kullanılmasına izin verilmiştir.

Bu bilgiye erişimi kolaylaştırmak ve her aracın aynı analizi uyguladığından emin olmak için, bir bean sınıfını analiz ederken kullanılması gereken

`java.beans.Introspector` sınıfı tanımlanmıştır. Bu sınıf, hedef bir bean sınıfını tanımlayan bir BeanInfo nesnesi elde etmeye izin verir.

Introspector (içgözlem) sınıfı, hedef sınıfın sınıf/üstsınıf zincirinde ilerler. Her seviyede, bean hakkında açık bilgi sağlayan uyumlu bir BeanInfo sınıfının olup olmadığını kontrol eder ve eğer bulursa, bu açık bilgiyi kullanır. Aksi halde, hedef sınıf hakkında bilgi edinmek için alt seviye yansıma API'sini kullanır ve davranışını anlayabilmek için de tasarım yapılarını kullanır.

3.4.6 İçgözlem ve Güvenlik

Java için önemli bir kavram olan güvenlik, JavaBean'lerde de karşımıza çıkar. Java'da güvenlik denildiğinde, büyük ölçüde bir uygulamanın güvenilir olup olmamasından bahsedilir. Güvenilir ugulamalar, tek başına Java uygulamalarını ve sayısal olarak imzalanmış appletleri içerir. Güvenilir olmayan uygulamalar ise, sayısal olarak imzalanmamış appletlerdir.

Güvenliğin, JavaBean'lerde ayrıca düşünülmesinin sebebi, içgözlemdir. JavaBean'ler tasarlanırken, kullanıcının sabit diski gibi diğer kaynaklarda olduğu gibi, bir bean'in iç durumu hakkındaki bilgilerin güvenilir olmayan uygulamalardan korunması gerektiğine karar verilmiştir. Bunun sebebi, güvenilir olmayan bir uygulamanın, bir bean'in bazı bölümlerini değiştirmesinin istenmemesidir. Güvenilir olmayan uygulamaların sadece, genel metodlara, özelliklere ve olaylara ulaşmasına izin verilir. Ancak bu bir problem yaratmaz, çünkü bunlar, bir bean'in dışarıya açılma yollarıdır.

Bir uygulama geliştirme aracı gibi, güvenilir bir uygulama ise, bir bean'in genel bilgilerinde sunulandan daha fazlasını bilmeye ihtiyaç duyar. Güvenilir uygulamalar söz konusu olduğunda, güvenlik sınırlamaları kaldırılır ve uygulamalara bean'in yapısına daha derin erişme izni verilir. Bu aşamada yine, alt-düzey ve üst-düzey API hizmetleri hatırlanabilir. Alt-düzey API'ler, güvenilir uygulamalar için ve üst-düzey API'ler ise, güvenilir olmayan uygulamalar için mevcuttur.

3.5 Kalıcılık

Bir bean'in iç durumunun, daha sonraki bir kullanım için saklanabilmesi gereklidir. Bir bean'i, kalıcı bir yerde saklama işlemine kalıcılık adı verilmiştir. Kalıcılık, Java Bean'ler için önemlidir, çünkü uygulama geliştiriciler, bu sayede, bir uygulamayı oluşturan bean'lere yaptıkları değişiklikleri saklama imkanını elde ederler. JavaBean API'si, bir bean'in iç durumunu kalıcı olarak saklayabilmek için gerekenleri sunmaktadır. Kalıcılık, geliştirme araçları açısından önemli olmanın yanında, veriye-yönelik bean'lerin daha sonraki bir geri yükleme için, kendilerini saklamalarına imkan vermektedir.

Bean'lerin yeniden yaratılması oldukça kolay bir işlemdir ancak bean'lerle ilgili önemli bir nokta da, herhangi bir kullanıcı tarafından, farklı bir durum için yerel uyarlama imkanını sağlamalarıdır. Tabii, geliştiricinin, bean'in yerel uyarlama sonucundaki durumunu saklanmak ve daha sonra tekrar yüklemek istemesi çok normaldir.

3.5.1 Kalıcılıkta Saklanması Gereken Elemanlar

Bir bean kalıcı hale getirildiğinde, daha sonra benzer görüntü ve davranışla tekrar elde edilebilmek için, iç durumunun belirli kısımlarını saklamalıdır. Normalde, bir bean tüm dışa açılan özelliklerinin kalıcı halini saklayacaktır. Ayrıca, özellikler aracılığı ile doğrudan erişilemeyen ek iç durum bilgisini de saklayabilir. Buna örnek olarak, bir bean yerel uyarlayıcı kullanarak yapılan ek tasarım seçimleri veya bean geliştirici tarafından yaratılan iç durum verilebilir.

Bir bean, diğer başka beanleri içerebilir. Bu durumda, kendi iç durumunun bir bölümü olarak bu beanleri saklamalıdır.

Bir bean, geçici bilgiyi takip etmek için üye değişkenleri kullanabilir. Ancak bu durumda, bu bilgiyi, kalıcı olarak saklamak anlamlı değildir. Bu bilgi, bir şekilde, çalışma anında yeniden elde edilmelidir. Yani, kalıcı hal bilgisi, sadece koşma-anı ortamından elde edilemeyen şeyleri içermelidir.

Kalıcı olarak saklanması gereken bilgilerin türüyle ilgili bir başka şaşırtıcı nokta, diğer bean'lere başvurulardır. Örnegin, taşıyıcı bean'ler, içerdikleri tüm bean'lere ait

başvuruları idare etmelidirler. Ancak, bean başvurularını saklamak ve tekrar elde etmek mümkün değildir, çünkü başvurular aslında geçici olan bellek işaretçileridir. Diğer bir deyişle, bellek sürekli değiştiğinden, bir bellek işaretçisinin bir sonraki yükleyişte anlamlı herhangi bir şeye işaret etme ihtimali yoktur. Yalnız, beanler arasındaki başvuruları saklayabilmek önemlidir. Bu durumda, JavaBean, kalıcılık üzerinden başvuru problemini halledemediğinde, sorumluluğu, beanlerin kullanıldığı uygulama veya uygulama geliştirme aracına yükler. Böylece, bir geliştirme aracı kullanarak bir grup bean'i yerel uyarlar, birbirine bağlar ve herşeyi saklarsanız, bean'ler JavaBean kalıcılık mekanizması kullanılarak saklanırlar ama birbirleri ile olan ilişkileri bir tür geliştirme aracına özel bir yöntem ile saklanır.

Yani, sonuç olarak, geçici bilgiler ve diğer beanlere olan başvurular dışında, bir bean'e ait verilerin çoğu kalıcı olarak saklanmalıdır. Bu noktada, JavaBean yapısının, bir bean'e ait hangi bilgileri kalıcı olarak saklayacağına nasıl karar verdiğine bakalım. JavaBean, tüm üye değişkenlerin kalıcı olarak saklanacağını kabul eder. Saklanması istenmeyen değişkenler içinse, Java'da tanımlı `transient` anahtar kelimesi kullanılır. Bu anahtar kelime, bir üye değişkenin, bir nesnenin kalıcı durumunun bir parçası olmadığını belirtir.

3.5.2 Kalıcılıkta Saklama İçin Zincir Kullanımı

Java'da kalıcılık, nesne düzeyinde gerçeklenmektedir. Bilgiyi belirli bir dosyaya saklamak ve dosyadan okumak yerine, Java, bir nesnenin içeriklerini bir zincire kaydettiği bir nesne ağacı üzerinden işler. Bu zincir, bir dosya veya bir saklama birimi olabilir. Program tekrar başlatıldığında, zincir yeniden okunabilir ve kalıcı bilgiyi içeren baştaki nesne ağacı yeniden oluşturulabilir.

Bir nesneyi, bir zincire yazma veya zincirden okuma sürecine serileştirme denir.

3.5.3 Kalıcılık İçin Gerekenler

Bir sınıfın serileştirilmesi için, iki nokta önemlidir. Birinci olarak, sınıfın `java.io.Serializable` veya `java.io.Externalizable` arayüzünü gerçeklemesi gereklidir.

`Serializable` arayüzü, gerçeklenmesi en kolay olanıdır, çünkü kullanılması gereken herhangi bir metod içermez. Sadece, serileştirme araçlarına, sınıfın serileştirilebileceğini gösteren bir işaret olarak düşünülebilir. `Externalizable` arayüzü ise, serileştirme sürecini denetlemek için, bazı metodların gerçeklenmesini gerektirir.

Bir sınıfı serileştirilebilir yapmak için gerekli ikinci kural ise, sınıfın, hiç bir parametre almayan varsayılan bir kurucusunun olmasıdır. Bunun gerekliliği şöyle açıklanabilir. Bir nesne geriserileştirilirken, bir önceki durumu oluşturulmaktadır. Bir nesnenin durumu, farklı kurucular ve nesnenin iç durumu üzerinde işlem yapan metodlar ile oluşturulmuş olabilir. Bu sırada hangi kurucunun kullanıldığına veya hangi metodların çağırıldığına ait bir kayıt olmadığı için, aynı sıralı olayları tekrar oluşturma şansımız yoktur. Bu yüzden, geri alındığında, Bean'in alanlarına sadece bir önceki duruma ait bilgiler alınabilir. Bu aşamada, varsayılan kurucu ile temel nesne yaratılır ve gerekli alanlar, alınan bilgilerle doldurulur.

3.5.4 Kalıcı JavaBean'ler

Kalıcılığı, bir bean açısından incelediğimizde, bazı farklı önemli noktalarla karşılaşabiliriz. Kalıcı Java Bean'ler, genellikle sadece bir başlangıç durumu olan nesnelerdir. Bir uygulamanın her yüklenişinde, Java bean'leri geriserileştirilir ve uygulamayı gerekli bileşenlerle tamamlarlar. Diğer bir deyişle, bir bean'i kullanan uygulama geliştirici, bean'i yükleyerek, geriserileştirme işlemini başlatır.

Çoğu bean'ler, kalıcı bir yapılandırmaya sahip olan bileşenler olarak ele alınırlar. Bir bean'in durumu, bean bir uygulama geliştirme aracında, uygulama geliştiricinin istediği şekilde yapılandırıldığında saklanır. Uygulama çalıştırıldığında, saklanan durum geri yüklenir ve bean bileşeni, sanki yaratılmış ve kurma komutları kullanılarak yapılandırılmış gibi kullanılır. Uygulama sona erinceye kadar, bean'in durumu geri saklanmaz.

Bean'ler, kendi ilk değer atamalarını kendileriyle beraber taşırlar. Bu, programcının başlatma ve yapılandırma kodunu yazma gereğini ortadan kaldırır. Programcının, sadece programın içinde, bileşeni diğerleriyle bağlaması gereklidir. Renkler, büyüklük ve diğer parametreleri ayarlamak yerine, bean, kendisini, geliştiricinin

IDE'sindeki kalıcı durum kümesinden yapılandırır. Bu, kalıcılığın getirdiği önemli bir kazançtır.

3.5.5 Bean'leri Serileştirme

Bir bean'in hayatında, serileştirme, bean'in nasıl görünmesi ve davranması gerektiğinin hafızasıdır. IDE'de belirlenen her bilgi kümesi, bean son uygulamada kullanıldığında geri getirilmek üzere, saklanır. Başlatma ve ilk değer atama sırasında, serileştirilmiş bir bean'in veya bir uygulama programının yapacağı çok az şey vardır. Herşey, nesneyi geriserileştirme işlemi ile yapılmaktadır. Yani bir sınıfta, sadece arayüzünü gerçeklemek ve bir varsayılan kurucu desteklemek, nesnenin alanlarını serileştirmeyi mümkün kılar. Bu nesnedeki tüm ilkeller serileştirilebilirler. Nesne olan tüm alanlar da, sınıfları `java.lang.Serializable` veya `java.lang.Externalizable` arayüzlerini gerçekliyorsa, serileştirilirler. Bu serileştirme işlemi, bir nesne ağacındaki tüm veri saklanana kadar devam eder. Aynı işlem, bean geri yüklendiğinde de geçerlidir. Nesne ağacındaki tüm veri, saklanan değerlere ayarlanır.

`java.lang.Externalizable` arayüzü, tekbaşına, bir nesnenin serileştirilmesini denetlemek istediğimizde kullanılır. Örneğin, serileştirilmeden önce, sıkıştırılması, şifrelenmesi veya başka nesnelerle senkronize edilmesi gerekebilir. Bu durumda, `Externalized` arayüzü gerçeklenir ve içerdiği iki genel metod kullanılır. Bu metodlar, bir nesneyi serileştirmek ve geriserileştirmek için kullanılır. Bu durumda, geliştirici, nesne verilerinin okuması ve yazılmasını denetlemekle sorumludur.

Bu durumda, `Serializable` arayüzü kullanılarak, serileştirilme ve geriserileştirilme işlemlerinin otomatik olarak yapıldığı ve `Externalizable` arayüzü gerçeklenerek ise, bu işlemlerin tamamen geliştiriciye bırakıldığı ve tüm kontrolün geliştiricide olduğu söylenebilir.

3.5.6 Sürüm Oluşturma

Kalıcılık ve serileştirme ile ilgili önemli bir nokta, nesne işlevselliğinin gelişimini içeren sürüm oluşturmadır. Bir nesne, bir sürümle saklandığı ve başka bir sürümle geri yüklendiğinde bir problemle karşılaşırız. Çünkü, bir nesneye ait farklı sürümler,

farklı durum bilgilerine sahip olacaklardır. Burada, bir nesneye ait farklı sürümler arasında uyumluluk sağlanması gereği ortaya çıkar.

Sürüm oluşturma ile ilgili yol göstermek için, Java'daki serileştirme kolaylıkları bazı kural kümeleri tanımlarlar. Bu kurallar sayesinde, farklı sürümler arasında uyumluluk sağlanabilir. Bu kurallar, yapılabilecek değişiklikleri sınırlandırarak, nesnelerin gelişimini belirlerler.

Java API'deki serileştirme mekanizması, aşağıda listelenen maddelerle sürüm oluşturmayı destekler :

- Farklı sanal makinalarda yürütülen, bir nesnenin farklı sürümleri arasında iletişimi destekler.
- Eski sürümlü nesneler tarafından yazılmış zincirleri okuyan nesneleri destekler.
- Eski sürümlü nesnelerin okumak istedikleri zincirleri yazan nesneleri destekler.
- Çoğu zaman verimli ve küçüktür.
- Zinciri yazarken kullanılan sürümü tanır ve ona uygun nesneleri yükler.

3.6 Yerel Uyarlama

Yerel uyarlama, tasarım ortamında, bir bean'in özelliklerinin düzenlenmesi ve değiştirilmesi demektir. Java bean'lerinin geliştirilmesinde önemli bir yere sahiptir, çünkü bean'lerin, görsel bir tasarım aracında nasıl birleşebileceğini ve kullanılacağını tanımlar. Diğer bir deyişle, yerel uyarlama, bean'in kullanıldığı durumda, uygun şekilde görünmesi ve davranması için, iç durumunun yapılandırılması demektir. [5]

Yerel uyarlama iki türlü olabilir:

Bir bean bir grup özelliği dışa açtığında, uygulama geliştirme aracı, bu özellikleri kullanarak, bir GUI (Grafik Kullanıcı Arabirimi) *özellik sayfası* hazırlar. Bu sayfada,

özellikler listelenir ve her bir özellik için bir *özellik düzenleyici* sağlanır. Kullanıcı, bu özellik sayfasını kullanarak, bean'in çeşitli özelliklerini günceller.

Bu tür bir özellik sayfası, daha çok basit veya orta büyüklükte bean'lerde kullanılmaya uygundur. Ancak, daha büyük veya karmaşık bean'ler söz konusu olduğunda, daha farklı ve karmaşık çeşitte bileşen yerel uyarlamasına ihtiyaç duyarız. Bu durumda, her Java bean'e, bean'in yerel uyarlamasını denetleyen bir yerel uyarlayıcı (customizer) sınıfın eşlik etmesine izin verilir. Bu yerel uyarlayıcı sınıf, hedef bir bean'i yerel uyarlamak için koşturulabilen bir AWT bileşeni olmalıdır. Bu sınıf, hedef bean'in yerel uyarlamasında kullanmak istediği herhangi bir GUI davranışını sağlayabilir.

3.6.1 Yerel Uyarlanan Bileşenlerin Saklanması

Bir kullanıcı, bir grup bileşenin davranışını yerel uyarladığında, uygulama geliştirme aracı, kalıcılık mekanizmasını kullanarak, bileşenlerin durumunu saklamalıdır. Uygulama koşturulduğunda, değiştirilen ve saklanan durum, bileşenlerin ilk değerlerini atamak için geri okunmalıdır.

3.6.2 Özellik Düzenleyiciler

Bir özellik düzenleyici, String veya Color gibi bir özelliğin değerini düzenlemek için bir kullanıcı arayüzüne izin veren bir sınıftır. Bean geliştiriciler, bean'lerinin bir parçası olarak sundukları yeni bir veri tipi için de yeni özellik düzenleyiciler sunabilirler.

Bean geliştiriciler, kendi özellik düzenleyicilerini tanımlamak için, JavaBean API'sinin sunduğu `PropertyEditor` arayüzünü gerçeklemelidirler. Geliştiricilerin özellik değerini okumak ve yazmak için kullanabilecekleri pek çok farklı yol vardır. Tüm özellik düzenleyicilerinin tüm seçenekleri desteklemesi gerekli değildir. En basit düzeyde düşünüldüğünde, bir özellik düzenleyici, bir değeri bir katar olarak okumayı veya yazmayı destekleyebilir. Daha ileri bir düzeyde düşünülürse, bir özellik düzenleyici, özelliğinin düzenlenmesi için açılabilen, karmaşık bazı davranışları içerebilen bir bileşen olabilir.

Bir özellik düzenleyici, bir özellik sayfasının bir bölümü olarak veya daha karmaşık bir bileşen yerel uyarlayıcısında bir alan olarak başlatılabilir. Eğer bir özellik düzenleyicisi, (bir katar, boyanmış bir dikdörtgen, özel düzenleyici bileşeni, vs. gibi) farklı şekillerde gösterilmeye izin veriyorsa, verilen bir özellik düzenleyicisinin hangi şekilde en iyi temsil edileceğine karar vermek, daha üst düzey bir aracın sorumluluğudur.

3.6.2.1 Özellik Düzenleyicilerin Seçimi

Bazı özellik düzenleyicileri, Java Bean koşma anının bir parçası olarak sunulurken, bazıları, bileşen geliştiriciler tarafından, bazıları ise, uygulama geliştirme araçları tarafından sunulur.

Mevcut bir Java veri tipi için, doğru özellik düzenleyicisinin seçimini sağlamak üzere, Java tipleri ve ilgili özellik düzenleyici sınıfları eşleştiren `java.beans.PropertyEditorManager` sınıfı sunulmuştur.

PropertyEditorManager sınıfı, özellik düzenleyicilerin kayıtlarını tutar ve verilen bir Java tipi için özellik düzenleyicisi görevi görecek bir sınıf kaydetmeye izin verir. Yerleşik, standart Java tipleri için tanımlı düzenleyiciler, kayıtlarda önceden tanımlanmıştır.

Bir Java tipi için, bir özellik düzenleyicisi seçmeye kalktığınızda, özellik yöneticisi, aşağıdaki işlemleri sırayla gerçekleştirerek, uygun bir özellik düzenleyicisi arar:

- İlk önce, bir özellik düzenleyicisinin açıkça kayıt edilip edilmediğine bakar.

- Eğer bulamazsa, verilen Java tipi ismini alır, sonuna "Editor" ekler ve elde ettiği isme sahip sınıfı arar. Yani, "foo.bah.Wombat" için, özellik düzenleyicisi görevi görmesi için, "foo.bah.WombatEditor" sınıfını arayacaktır.

- Eğer bunu da bulamazsa, hiyerarşik yapıda verilen isimdeki son bileşeni alır, sonuna "Editor" ekler ve elde ettiği isme sahip sınıfı, paketlerin arama listesinde (varsayılan olarak java.beans.editors) arar. Örnek olarak, "foo.bah.Wombat" için, "java.beans.editors.WombatEditor"ü arayacaktır. Uygulama geliştirme araçları, paket arama yolunu, örneğin kendi sağladıkları bir paketle değiştirebilirler.

Normalde, yeni tipler tanımlayan ve özellik düzenleyicileri sağlamak isteyen bileşen yazarları, basitçe, adı temel tip adı + "Editor" olan bir düzenleyici sınıfı oluşturmalıdırlar.

Uygulama geliştirme araçları, temel Java tipleri için, JavaBean'lerle beraber sunulan varsayılan özellik düzenleyici kümelerini kullanabilir veya kendi özellik düzenleyici kümelerini tanımlayabilirler.

3.6.2.2 Değişiklikleri Bildirme

Bir özellik düzenleyici bir değişiklik yaptığında, bir "PropertyChange" olayı ateşlemelidir. Bu sayede, üst düzeydeki yazılımın, değişikliği farketmesine, yeni değeri almasına ve hedef bileşen üzerinde doğru setXXX çağrısını yapmalıdır.

3.6.2.3 Başlangıç Nesnesini Değiştirmeme

Özellik düzenleyicilere, bir özelliğin mevcut değerini içeren bir nesne verilir. Ancak bir özellik düzenleyicisi, bu ilk nesneyi doğrudan değiştirmemelidir. Yerine, kullanıcı girişine bağlı olarak, özelliğin değişen durumunu göstermek için yeni bir nesne yaratmalıdır. Özellik düzenleyici bir "PropertyChange" olayı ateşlediğinde, üst düzey yazılım, değişikliği ferkedecek, yeni nesneyi alacak ve yeni değeri ayarlamak için, hedef bileşen üzerinde uygun özellik belirleyici (setter) metod çağrısını yapacaktır.

Bazı özellik sayfaları, her değiştirilen özelliği, hemen hedef bileşene uygular. Bazı özellik sayfaları veya yerel uyarlayıcılar ise, kullanıcıdan bir doğrulama almadan, değişiklikleri geciktirmeyi tercih edebilirler.

3.6.2.4 Dolaylı Özellik Değişimleri

Bazı bileşenlerde, bir özellik değerinin ayarlanması, başka özellik değerlerinin değişmesine neden olabilir. Eğer bu, özellik sayfası açıkken meydana gelirse, kullanıcı, özellik sayfasının tüm yeni değerlerle güncellenmesini isteyecektir. Bu yüzden, her özellik değişiminden sonra, bir özellik sayfası yöneticisinin, hedef bean'le ilgili tüm özellikleri tekrardan okuması tavsiye edilir. Ancak sürekli ekranın

yeniden çizilmesinden sakınmak için, her yeni özellik değeri, bir önceki değeriyle karşılaştırılmalı, eğer değer değiştiyse, özellik düzenleyicileri güncellenmelidir.

3.6.3 Yerel Uyarlayıcılar

Çoğu basit bileşeni yerel uyarlamak için, özellik sayfalarının kullanımı yeterlidir. Ancak yapılandırılmak üzere birden fazla şekil ve olasılık sunan büyük ve karmaşık bileşenler için, bileşen geliştiricinin, bileşeni yerel uyarlamak için özel bir sınıf tanımlaması daha doğru olabilir. Bu sınıflara, yerel uyarlayıcılar denir.

Bir özellik sayfası yerine, bir yerel uyarlayıcı kullanmanın birçok sebebi vardır. Tasarlanan bean'in karmaşıklaşması, sunulan özelliklerin sayısının artması demektir. Bean'i oluşturan sınıfın miras aldığı sınıfın özellikleri de eklenince, özellik sayfaları, IDE'de tanımlı bulunan yapıda özellik düzenleyicilerle dolacaktır. Bu, hem görünüşü kötüleştirir hem de kullanışı zorlaştırır, çünkü kullanıcı, doldurulacak bir çok alandan oluşan bir özellik sayfasıyla karşılaşır. Bean'in karmaşıklaşmasından ayrı olarak, bean'in tümü veya bir bölümünün, kullanıcı için daha anlaşılabilir bir şekilde sunulması gerekebilir. Bu tür durumlarda, özellik sayfasının kullanımı zorlaşır, çünkü özellik sayfasının denetimi IDE'dedir. IDE'nin de bean hakkında bir ön bilgisi olmadığından, uygun davranışları sergileyemez. Bu aşamada, bean geliştirici, bir yerel uyarlayıcı kullanarak, bean'in düzenleme sürecine ait denetimi, IDE'den almaktadır.

Bir bean yerel uyarlayıcısı, bean'in tümünü veya bir kısmını düzenlemek için, tam erişim hakkı olan bir düzenleyicidir. Çoğunlukla, bir yerel uyarlayıcı, bean'le aynı pakette yer alır. Bu paket üyeliği, bir bean'in IDE tarafından ulaşılamayan korumalı işlevlerini değiştirebilmek için önemlidir. Yerel uyarlayıcılar, ayrıca düzenlenebilir bir görüntü sunmaları açısından da önemlidirler.

Her yerel uyarlayıcı, doğrudan veya dolaylı olarak java.awt.Component'ı miras almalıdır. Ayrıca, java.beans.Customizer arayüzünü de gerçeklemelidir.

Normalde, her yerel uyarlayıcı ayrı bir AWT diyalog penceresinde koşturulur. Yerel uyarlayıcı, kendini nasıl sunacağına tamamen kendi karar verir. Kullanıcı, yerel uyarlama sürecinde farklı seviyelerde ilerlerken, görünüşünü tekrar çizebilir. Ayrıca,

bir yerel uyarlayıcı, hedef veri değerlerini düzenlemek için görüntüsüne, özellik düzenleyiciler dahil edebilir.

Her yerel uyarlayıcıya, kurucu değişkeni olarak, düzenlemesi için bir hedef nesne verilecektir.

Değişiklikleri adım adım ilerledikçe uygulamak veya saklayarak, belirli noktalarda nesneye uygulamak tamamen yerel uyarlayıcıya bağlıdır. Ancak, hedef bean'in görüntüsünün, istenen değişiklikleri yansıtmak için mümkün olduğu kadar çabuk güncellenmesi tavsiye edilir.

Kendi yerel uyarlayıcısını sunmak isteyen bir bean, kendi BeanInfo sınıfını da sunmalıdır. Geliştirme araçları, bu BeanInfo sınıfını kullanarak, yerel uyarlayıcı sınıfını nereden elde edecekleri bilgisini alırlar.

3.7 JavaBean'lerin Paketlenmesi

JavaBean'ler, diğer araçlar ve uygulamalar tarafından kullanılabilmek için, JavaSoft tarafından tanımlanan düzende paketlenmelidirler. Ancak JavaSoft, bir uygulamanın, JavaBean'leri saklarken kullanacağı düzen hakkında bir şey belirlememiştir. Bean'ler temelde, JAR dosyaları şeklinde dağıtılsalar da, bir uygulama içinde kullanılırken yeniden paketlenebilirler.

3.7.1 Jar Dosyalarına Genel Bir Bakış

JavaBean'ler, Jdk1.1'de desteklenmeye başlayan yeni bir teknoloji olan, JAR dosyaları şeklinde paketlenir ve dağıtılırlar. JAR dosyaları, sınıf dosyalarını, serileştirilmiş nesneleri, görüntüleri, yardım dosyalarını ve benzer kaynak dosyaları bir arada tutmak için kullanılırlar.

Bir JAR dosyası, seçeneğe bağlı olarak, bir bildirim (manifest) dosyasına da sahip olabilen, ZIP düzeninde bir arşiv dosyasıdır. Bean içeren bir JAR dosyasının, bu bean'leri tanımlayan bir bildirim'i olmalıdır. Bildirim dosyası, JAR dosyasının içeriğini tanımlayan ek bilgi sunar.

Bir JAR dosyası, bir veya birden fazla bean içerebilir.

3.7.2 Jar Dosyalarının İçeriği

Bir JavaBean JAR dosyası, paketlenmiş bir veya birden fazla JavaBean'i tanımlayan öğeler içerir. Kolaylık sağlaması için, JAR dosyası öğelerine, '/' işareti ile ayrılmış isimler verilir.

Bir JavaBean JAR dosyası, aşağıda tanımlarının listesi verilmiş öğeleri içerebilir :

- Bir bean'in davranışını belirleyen bir sınıf dosyası kümesi. Bu öğelerin, .class uzantılı isimleri olmalıdır.
- Seçime bağlı olarak, bir bean'i ilklendirmek için kullanılan serileştirilmiş bir prototip. Bu öğeler, .ser uzantılı isimlere sahip olmalıdır.
- JavaBean için dökümantasyon sağlayacak, HTML formatında, seçime bağlı yardım dosyaları.
- Bean'in kendisini yerelleştirmek için kullanacağı, seçime bağlı, uluslararasılaştırma (internationalization) bilgisi.
- JavaBean tarafından kullanılacak, ses, resim vb. kaynak dosyaları.

3.7.3 Bean İsimleri

Bir bean adı, ' . ' ile ayrılmış isimlerin sıralanması ile elde edilen bir katardır.

Basit bir şekilde, sınıflardan oluşturulan bean'ler, karşı düşen sınıfla aynı isme sahiptirler. Bunlar, JAR öğelerine dönüştürülürken, her ' . ' , bir ' / ' ile değiştirilir ve sonuna '.class' eklenir. Yani bir 'x.y.z' sınıfı, bir JAR öğesine karşı düşürülürken, 'x/y/z.class' şeklini alır.

Serileştirilmiş prototipleri temel alan bean'lerin de isimleri, sınıf isimlerine benzer ve ' . ' larla ayrılırlar. Bunlar JAR öğelerine dönüştürülürken, her ' . ' , bir ' / ' ile değiştirilir ve sonuna '.ser' eklenir. Böylece 'a.b.c' isimli serileştirilmiş bir bean, 'a/b/c.ser' şeklinde bir JAR öğesine dönüştürülür.

Bean isimleri de, sınıf isimleri gibi, belirli bir sınıf yükleyici içinde, tek olmalıdır.

Bir veya birden fazla bean içeren bir JAR dosyası, bu arşivde yer alan öğelerin hangilerinin bir bean'e karşı düştüğünü belirten bir bildirim dosyasına sahip olmalıdır. '.class' ile biten bir kayıt, ilgili sınıfın bir bean olduğunu, '.ser' ile biten bir kayıt ise, serileştirilmiş bir bean'in içerildiğini gösterir.

3.7.4 Bildirim Dosyasının Düzeni

Bir JAR arşivi, JAR dosya düzeninin bir parçası olarak, içeriklerini tanımlayan bir bildirim dosyası içerebilir.

Bir JAR arşivinde yer alan bildirim dosyası, arşivin içeriğinin belirli bazı kısımları hakkında bilgi sunar. Birbirinden boş satırlarla ayrılan bölümler dizisidir. Her bölüm, bir veya birden fazla başlık içerir. Başlıkların her biri, <etiket>: <değer> şeklindedir. Arşivde yer alan dosyalarla ilgili bilgi sağlayan bölümler, <etiket>i 'Name' ve <değer>i de tanımlanan dosyanın bağıl adı olan bir başlık içermelidirler.

Bir bildirim dosyası, JAR arşivinde yer alan bean'leri tanımlayan bilgi içerebilir. Eğer JAR arşivinde yer alan bir dosya, bir bean'se, bildirim dosyasında ona karşı düşen bölümde, <etiket>i 'Java-Bean' ve <değer>i 'True' (büyük, küçük harf önemli değildir) olan bir başlık olmalıdır.

Uygulamada yer alan örnek bir bildirim dosyasının içeriği aşağıda verilmiştir.

```
Name: GanttChart.class
Java-Bean: True
```

Bir JAR dosyası oluşturmak için Jdk1.1'de yer alan "jar" komutu kullanılabilir. Bu komutun kullanımına örnek olarak, yine uygulamada yer alan GanttChart bean'inin paketlenmesi verilebilir. Bu komutta yer alan manifest.tmp dosyası, herhangi bir metin editörüyle, yukarıdaki örnek satırları içerecek şekilde oluşturulabilir.

```
jar cfm GanttChart.jar manifest.tmp GanttChart.class Day.class
```

3.7.5 JavaBean'lerle Kullanılan Bildirim Başlıkları

Java bean'lerle kullanılmak üzere üç başlık etiketi tanımlanmıştır :

- Java-Bean : Bu başlık etiketi, Java bean'leri tanımlayan JAR öğelerini belirlemek için kullanılır. Bu en çok kullanılan bildirim dosyası girişidir. Yukarıda detaylı açıklama verilmiştir.

- Depends-On : Bir JAR öğesine ait bildirim bölümü, seçime bağlı olmak üzere, bağlı olduğu diğer JAR öğelerini belirlemek üzere, 'Depends-On' etiketini kullanabilir. Bir bildirim bölümü, sıfır, bir veya birden fazla 'Depends-On' satırı içerebilir. Eğer satır sayısı sıfırsa, bağlılık durumu bilinmiyor demektir. Verilen bir 'Depends-On' satırı ise, sıfır, bir veya birden fazla bağlılık belirtebilir. Eğer belirtilen bağlılık sayısı sıfırsa, bu bean'in bağlılığı yok demektir.

- Design-Time-Only : Bir JAR öğesine ait bildirim bölümü, seçime bağlı olmak üzere, verilen öğenin sadece tasarım zamanında gerekli olup olmadığını belirlemek üzere, 'Design-Time-Only' etiketini kullanabilir. Bu, geliştirme araçlarının, yapılandırılan bir uygulamanın bir parçası olan bir bean'i paketleme sırasında, verilen bir öğeyi dışlamasına imkan tanır.

4. "JAVA BEAN"LERİ VE BİR VERİTABANINA ERİŞİM UYGULAMASI

4.1 Tanımlama

Bu projede, İTÜ Proje Yönetim Merkezi için, İTÜ Bilgi İşlem Merkezi tarafından geliştirilen bir otomasyon yazılımının bir bölümü gerçekleştirilmektedir. Bu çalışmada amaç, bir veritabanında tutulan, inşaat projelerine ait süre ve maliyet bilgilerinin bu veritabanından okunması ve kullanıcının web üzerinden bu bilgilerin grafiksel sunumuna erişiminin ve belirli kriterlerle farklı gösterimler elde ederek, projelere ayrılacak maliyetler konusunda karar vermesinin sağlanmasıdır.

Her inşaat projesine ait bilgiler, Proje Yönetim Merkezi'nde çalışan görevlilerce, hazırlanan veri giriş ekranları üzerinden girilecek ve Microsoft SQL Server veritabanı üzerinde tutulacaktır. Ayrıca projelere ait malzeme ve tanımlanan iş kalemleri ile ilgili maliyet bilgileri, başlangıç-bitiş ilişkileri kurularak, zaman planlaması yapmak üzere kullanılan Primavera adlı bir programa verilecektir. Bu çalışma, giriş bilgilerini oluşturan veritabanı tabloları olarak, Primavera'nın tabular düzende verdiği çıkışları kullanmaktadır.

Bir ekranda bir inşaat projesine ait Gantt diyagramı verilir ve kullanıcıdan alınan maliyet bilgisi, bir projeye ne kadar maliyet aktarılacağı bilgisi, kullanılarak o projenin ne kadar ilerleyebileceği Gantt diyagramı üzerinde belirtilir. Böylece farklı projelere girilen maliyet bilgilerinden alınan sonuçlar incelenerek, hangi projeye ne kadar para aktarılacağına karar verilebilmesi planlanmaktadır.

Kullanıcı girişi, veritabanından okuma ve gantt diyagramı çizdirme bölümleri birer JavaBean bileşeni olarak tasarlanmıştır.

4.2 Gerçekleme

Çalışma süresince gerçeklenen üç bean, bir uygulama geliştirme aracı kullanılarak, yerel uyarlanır, birbirleriyle ilişkilendirilir ve ortaya çıkan uygulama, hem geliştirme ortamında, hem de istenen hali aldıktan sonra, applet haline getirilip, bir inceleyici (browser) içinde, çalıştırılabilir.

Bu bölümde, bean'ler sırayla, geliştirme aracında yerel uyarlama ve diğer bean'lerle ilişkilendirme yapılabilmesi için içerdikleri kod kısımları bakımından incelenecekler dir. Daha sonra bu yerel uyarlama ve ilişkilendirme için bir örnek verilecek ve Proje Yönetim Merkezi'nde kullanılması planlanan uygulamanın nasıl oluşturulacağı ele alınacaktır. Bu çalışma boyunca, hem geliştirilen bean'lerin, tek tek istenen davranışı sağlayıp sağlamadığı, hem de bir arada kullanılarak hedeflenen uygulamanın nasıl elde edileceğini görebilmek için, Javasoft (Sun) tarafından sunulan bir uygulama geliştirme aracı olan, BeanBox kullanılmıştır.

4.2.1 Kullanıcı Girişi Bean'i

Bu bean, öncelikle, kullanıcıdan maliyet bilgi girişini alabilmek ve bunu gantt diyagramı bean'ine aktarmakla görevli bileşendir. Bu maliyet bilgisi, gantt diyagramı bean'i tarafından, çizimin güncellenmesi sırasında kullanılacaktır.

Kullanıcı girişi bean'i (CostBox adı verilmiştir), temelde özel bir textfield bileşeni olarak düşünülebilir. Sadece tamsayı değerler alması tasarlanmıştır. Bu amaçla, oluşturulan sınıf, java.awt.TextField'den miras alır ve kendi bazı yenilikler ekler.

Kullanıcı girişi bean'i, uygulama geliştirme aracında yerel uyarlanmasını sağlamak üzere tanımlanmış bazı özellikler içerir. Aslında, kodda doğrudan tanımlanan özellik sadece kullanıcıdan aldığı değeri tutan `value` özelliğidir. Uygulama geliştirme aracındaki özellik sayfasında görünen diğer tüm özellikler, bir üst sınıf olan Textfield sınıfında tanımlı özelliklerdir.

Bu özelliğin, uygulama geliştirme araçları tarafından anlaşılabilmesini sağlayan, tanımlı erişim metodlarıdır. Bu metodlar temelde aşağıda verilen yapıdadır.

```
public int getValue()
{
}

public void setValue(int value)
{
}
```

Bu bean'deki kilit nokta, `value` özelliğinin bağlı bir özellik olarak tanımlanmış olmasıdır. Bu sayede, gantt diyagramı bean'i ile arasında tanımlanan ilişki ile, gantt diyagramı bean'inden bu değeri okumak ve bu özellikte yapılan her değişikliği takip etmek mümkün olmaktadır. Aşağıda, bu özelliğin bağlı özellik olarak tanımlanması için yazılan kod satırları ve açıklamaları verilmiştir :

```
PropertyChangeSupport pcs_;
```

Öncelikle, yardımcı sınıfa ait bir örnek oluşturulur. Bu örnek, kodun ilerleyen kısımlarında, bağlı özelliğe ait dinleyiciler listesini tutmak ve özelliğin değerinde bir değişiklik olduğunda, listedeki bütün dinleyicilere bildirmek üzere kullanılmaktadır.

```
pcs_ = new PropertyChangeSupport(this);
```

Ayrıca, yukarıdaki ilklendirme satırı, bean'in kurucu (constructor) metodunda yer almaktadır ve o bean'in bağlı özelliğin tanımında kaynak bean olduğu anlatılmaktadır.

Bundan sonraki adım olarak, bu bağlı özelliğe kendini dinleyici olarak kayıt ettirmek veya kaydını silmek isteyen sınıflar tarafından kullanılmak üzere birer metod oluşturulmalıdır.

```
public void addPropertyChangeListener(PropertyChangeListener
listener)
{
      pcs_.addPropertyChangeListener(listener);
}

public void removePropertyChangeListener(PropertyChangeListener
listener)
{
      pcs_.removePropertyChangeListener(listener);
}
```

Yukarıda görüldüğü gibi, bu metodların gerçeklenmesinde yaratılan yardımcı sınıf kullanılmıştır. Tanımlanan bu 'add' ve 'remove' metodları ile, dinleyici listesi ile ilgili bütün sorumluluklar bu yardımcı sınıfa yüklenmektedir ve bu da, bean geliştiricisinin işini büyük ölçüde kolaylaştırmaktadır.

Bağlı bean'le ilgili olarak yapılması gereken son şey ise, bu özelliğin değerinde bir değişme olduğunda, kayıtlı dinleyicilere, bu değişimi ve özelliğin yeni değerini bildirmek üzere bir olay ateşlemektir. Bu olay ateşleme çağrısı, özelliğin erişim metodlarından, belirleme (setter) metodunda yapılmalıdır. Bu amaçla, bu erişim metoduna, özelliğin değerinde istenen değişiklik yapıldıktan sonra, gerekli çağrıyı yapmak üzere, aşağıdaki kod satırı eklenir.

```
pcs_.firePropertyChange("value", new Integer(oldValue), new
Integer(newValue));
```

Kullanıcı girişi bean'ine ait Java kodu aşağıda verilmiştir:

```
import java.io.*;

import java.awt.*;

import java.awt.event.*;

import java.lang.*;

import java.beans.*;

//Kullan c dan maliyet bilgisini almak icin haz rlanm , sadece
//tamsay de erler alabilen bir textfield.

public class CostBox extends TextField implements Serializable

{

      //CostBox'ta yeralan bir önceki de er

      int prev_;

      PropertyChangeSupport pcs_;

      //Varsay lan kurucu

      public CostBox()

      {

            super();

            enableEvents(java.awt.AWTEvent.KEY_EVENT_MASK);
```

```
            pcs_ = new PropertyChangeSupport(this);

            prev_ = 0;

      }

      //Di er nesnelerin ba l  özelliklere kay t olmas na izin
      //vermek üzere tan mlanan metodlar

   public void addPropertyChangeListener(PropertyChangeListener
listener)

      {

            pcs_.addPropertyChangeListener(listener);

      }

      public void removePropertyChangeListener
(PropertyChangeListener listener)

      {

            pcs_.removePropertyChangeListener(listener);

      }

      //Bu nesnenin de erini tamsay  olarak döndür.

      //Eger de eri elde edemiyorsak, s f r döndür.

      public int getValue()

      {

            int i;

            try {

                  i = Integer.parseInt(getText());

                  return i;

            }

            catch (NumberFormatException ex) {

                  return 0;

            }

      }

      public void setValue(int value)

      {

            int oldValue = prev_;
```

```
            setText(Long.toString(value));

            int newValue = getValue();


            //Bu ca r yla, tüm dinleyicilere özellik de i im
            //olaylar  gönderilir.

            pcs_.firePropertyChange("value", new Integer(oldValue),
new Integer(newValue));

            prev_ = newValue;

      }

      //Nümerik olmayan ve backspace'den ba ka karakterleri alma.

      public void processKeyEvent(KeyEvent event)

      {

            char c = event.getKeyChar();

            if ((c >= '0' && c <= '9') ||

                  c == KeyEvent.VK_DELETE ||

                  c == KeyEvent.VK_BACK_SPACE) {

                        return;

      }

            //E er 'Enter'a bas l rsa, de eri al.

            if (c == KeyEvent.VK_ENTER) {

                  int i = getValue();

                  setValue(i);

                  return;

            }

            //Di er tüm karakterleri at.

            event.consume();

      }

}
```

4.2.2 Veritabanı Erişim Bean'i

Bu bean'in görevi, kısaca veritabanına bağlantı kurmak, istenen bilgileri alıp, bu bilgileri kendisinden öğrenmek isteyen başka bir bileşene aktarmak için gerekli hazırlıkları yapmaktır.

Bu tanımı biraz daha açıklarsak, bean, öncelikle, bağlantı kuracağı veritabanı ile ilgili bazı bilgileri kullanıcıdan almak için gerekli özellik tanımlarını içerir ve kullanıcıya, bean'in özellik sayfasında birer özellik düzenleyici olarak sunar. Daha sonra, bu özellik bilgilerini kullanarak, istenen veritabanı bağlantısını kurar. Ardından, Primavera adlı programın oluşturduğu tabloya ulaşır. Bu tablo düzeni belirli olduğu için, tablo ve sütunları ile ilgili bilgilerin yerel uyarlanabilir olmasına gerek duyulmamıştır. Bu tabloya erişim ve yapılan bir sorgu sonucunda, gantt diyagramı bean'inin ihtiyaç duyduğu verileri elde edilir. Bu veriler, yapılan sorgu sonucunda elde edilen bir sonuç kümesi şeklindedir. Diğer bileşenlerin, bu sonuç kümesine erişmesi ve herhangi bir hücrede bir değişiklik olduğunda bunu takip edebilmesi için, sonuç kümesinin her bir sütunu birer bağlı ve indeksli özellik olarak tasarlanmıştır.

Bu uygulamada, gantt bean'i de aynı sayıda veri dizisine ihtiyaç duyduğu için, bu özelliklere bir dinleyici olarak kendini kaydettirir. Bu konudaki detaylar, daha ileride ve gantt diyagramı bean'inin açıklamalarında ele alınacaktır.

Proje Yönetim Merkezi'nde, bu uygulama sırasında kullanılacak veritabanı MS SQL Server olarak belirlenmiştir.

Bir Java appletinden, bir veritabanına erişim için, ilgili veritabanına özel ve Java ile yazılmış bir sürücünün kullanımı gereklidir. Bu sürücü sayesinde, herhangi bir web sunucuda bulunan ve bir veritabanına erişim işlemini içeren bir applete çağrı yapıldığında, applet'e ait bytekod'la beraber, gerekli sürücü yazılımı da istemci makinaya yüklenir ve böylece, kullanıcı tarafından özel bir tanım yapılmasına gerek kalmaz.

Eğer veritabanına özel bir sürücü yoksa, Javasoft'un JDK1.1 ile birlikte sunduğu Jdbc-Odbc köprüsü kullanılabilir. Yalnız bu durumda, appletin kodunda kullanılan 'veri kaynağı adı' (data source name) terimine karşı düşen veritabanı tanımı, kullanıcı

makinada, ODBC ayarlarında tanımlanmalıdır. Bu uygulama, Internet mantığında anlamsız olduğu için, Jdbc-Odbc köprüsü, yalnız Java uygulamalarında, sadece bir Intranet içinde kullanıma açık olan appletlerde veya servletlerde kullanılmaktadır.

Bu çalışmada hedeflenen, bean'lerin uygun şekilde yerel uyarlanıp, birbiriyle ilişkilendirilmesinden sonra elde edilen applete, sadece yönetici konumundaki birkaç kişinin ulaşması olduğu için, kullanıcı tarafında bir tanımlama gerekli olması herhangi bir problem teşkil etmemektedir. Bu yüzden, veritabanı bean'inin geliştirilmesinde Jdbc-Odbc köprüsü kullanılmıştır.

Veritabanına bağlantıyı kurarken kullandığımız özellikleri incelersek:

Veritabanına bağlantıyı tanımlamak için üç tane özellik tanımlanması uygun görülmüştür. Bunlardan birincisi, kullanıcı bilgisayarında tanımlanacak veri kaynağı adıdır. Microsoft SQL Server'da tanımlanan ve bean kodunda, veritabanına bağlantıyı kurma aşamasında kullanılan kullanıcı adı ve şifre de birer özellik olarak tanımlanmıştır. Bu üç özellik de, String tipinden tanımlanmışlardır. Uygulama geliştirici (bean'leri yerel uyarlayan kişi), veritabanı ile ilgili bu kullanıcı adı ve şifre bilgilerini ve kullanıcı tarafında tanımlanması gerekli veri kaynağı adını belirlemeli ve veritabanı bean'ini, bu verilerle doğru bir şekilde yerel uyarlanmalıdır. Ayrıca applet'e bağlanması öngörülen makinalarda, doğru Odbc ayarları yapılarak, doğru veritabanına erişim için, bean'in yerel uyarlanmasında kullanılan veri kaynağı adı kullanılmalıdır.

Bu üç özellik sadece bu bean içinde tanımlılardır. Yani değişimlerini izlemek isteyen başka bileşenler olması anlamlı görülmemiştir. Bu yüzden sadece erişim metodları ile özellik sayfasında yer almaları sağlanmış olur. Çok genel yapıda birer alıcı (getter) ve belirleyici (setter) erişim metodları vardır. Bu özelliklere bean'de 'dsn', 'uname' ve 'passwd' isimleri verilmiştir. Bu özelliklerden birine ait metodlar aşağıda verilmiştir.

```
public void setDsn(String dsn_)
{
      dsn = dsn_;
}

public String getDsn()
```

```
{
        return dsn;
}
```

Bu üç özellikten başka, Gantt bean'ine aktarılacak özellikler de vardır. Bu özellikler veritabanında yürütülen sorgu sonucunda elde edilen sonuç kümesinden oluşturulduğu için, kullanıcıya, bunlar için birer düzenleyici sunmak gerekli görülmemiştir. Bu yüzden, bu özelliklerin sadece belirleyici erişim metodları tanımlanmıştır. Bu aktarılacak özellikler, bağlı oldukları için, değişikleri dinlemek isteyen diğer bileşenlerin kayıtlarını takip etmek için gerekli dinleyici listesine bir ekleme ve bir de silme metodu mevcuttur. Ayrıca bir değişim olduğunda, ilgili özelliğin belirleyici metodunda yine değişim olayı ateşlenmelidir. Yine bu dinleyici listesi tutma ve olay ateşleme sırasında kullanılmak üzere `PropertyChangeSupport` yardımcı sınıfına ait oluşturulan örnek kullanılır.

Veritabanından yapılan sorgu sonucunda elde edilen sonuç kümesi, Gantt bean'e aktarılacak düzene getirilmelidir. Bu aktarım için bağlı ve indeksli özellikler seçilmiştir. Sonuç kümesinin her sütunu bir diziye karşı düşürülür ve bu diziler birer indeksli özellik olarak tanımlanır. Yalnız bu sırada dikkat edilmesi gereken bir nokta vardır. Dizilerin uzunluklarının, yaratılma sırasında verilmesi gerektiğinden, sonuç kümesi, öncelikle satır sayısını elde etmek için bir kere taranmalıdır, çünkü satır sayısını elde etmemizi sağlayacak bir metod, Java'da mevcut değildir. Bu yaklaşımda, sonuç kümesinin iki kere taranması gerekmektedir ve soruna sebep olan da bu işlemdir. Java, sonuç kümesinin ilk satırına bir işaretçi sunar ve her satır okunup işlendikçe, bu işaretçiyi bir ilerletme için bir metod tanımlıdır. Tekrar geri gitme veya işaretçiyi, istenen herhangi bir satırı gösterecek şekilde ayarlamak mümkün değildir. (Bu tür yeniliklerin Jdk1.2 ile sağlanacağı söylenmektedir.) Yani bir sonuç kümesi, sadece baştan sona ve sadece bir kere taranabilir.

Bu sorunun üstesinden gelebilmek için, programın geliştirilmesi sırasında vektörlerden yararlanılmıştır. Vektörler, başlangıçta uzunlukları sınırlandırılmayan diziler gibi düşünülebilecek nesnelerdir. Yaratılmaları sırasında, ilk uzunlukları ve kapasiteleri dolduğunda uygulanacak aktarım miktarı bilgileri verilir. Kullanıldıkları uygulamaya göre, bu parametreler en optimum çalışmayı sağlayacak şekilde ayarlanmalıdır. Bean'in kodunda, sonuç kümesinin sütunları önce, vektörlerde

saklanır ve bu sırada sonuç kümesinin satır sayısı, yani kullanılacak dizilerin uzunlukları belirlenir. Daha sonra, bu vektörler, bu eleman sayısı bilgisine göre yaratılan dizilere kopyalanırlar.

Veritabanından okunan ve diziler halinde dinleyici (Gantt diyagramı) bean'e aktarılan veriler, bir projede yer alan her iş kalemine ait id, isim, başlangıç tarihi, bitiş tarihi ve birim maliyet bilgileridir. Bunlardan başka, dizilerin eleman sayısı da bir tamsayı olarak diğer bean'e sunulmaktadır. İndeksli özelliklerden birine, iş kalemi ismi, ait belirleyici metod aşağıda verilmiştir.

```
public void setIsim(String[] isim)
{
      String[] oldValue = isimilk;
      pcs_.firePropertyChange("isim", oldValue, isim);
      isimilk = isim;
}
```

Dinleyici Gantt bean'i, kendini kayıt ettirdiği veritabanı bean'indeki birden fazla özelliğin değişimini takip ettiği için, bağlı özelliklerin kaynağı olan veritabanı bean'inin, olay değişim özelliğini ateşlerken bir parametre olarak kullandığı özellik adı bilgisi önem kazanmaktadır. Gantt bean'i tek bir propertyChange metodunda, kendisine sunulan tüm özellikleri, özellik isimlerini okuyarak ve kontrol ederek, gerekli dizilere ve değişkene alır.

4.2.3 Gantt Diyagramı Bean'i

Bu bean'in görevi, Veritabanı bean'inden aldığı sonuç kümesi sütunlarına karşı düzen dizileri ve kullanıcı girişi bean'inden aldığı maliyet bilgisini kullanarak, projeye ait gantt diyagramının istenen bölümünü çizmektir.

Gantt diyagramı bean'i, temelde bir Canvas ve bir ScrollBar (kaydırma çubuğu) nesnelerinden oluşan bir taşıyıcı olarak tanımlanmıştır.

Bu bean, kullanıcı girişi bean'indeki 'value' özelliğinde ve veritabanı bean'indeki indeksli özellikler ve bu dizilerin eleman sayısını tutan tamsayı özelliğinde meydana gelecek değişiklikleri dinlemek istediği için, `PropertyChangeListener` arayüzünü gerçeklemektedir. Gantt diyagramı bean'i, kullanıcı girişi bean'indeki, 'value' özelliği

ile bağlanmak üzere, maliyet bilgisini saklamak için 'cost' isimli bir özellik tanımı içerir. Veritabanı bean'inden gelecek dizileri ve bu dizilerin eleman sayısını tutan tamsayı bilgileri için de, uygun düzende birer nesne tanımlıdır ve içerdiği `PropertyChange` metodunda, tek tek bu özellikleri kontrol edip, ilgili nesneye aktarır.

Veritabanı bean'inde, bir indeksli özelliğe ait bir değişim olayını ateşleyen kod ile, Gantt diyagramı bean'inde, bu özelliğe ait uygun işlemi gerçekleştirebilmek için, özellik ismini karşılaştıran kod aşağıda verilmiştir.

Veritabanı bean'i (yukarıda verilen setIsim metodunda yer alan satır):

```
pcs_.firePropertyChange("isim", oldValue, isim);
```

Gantt diyagramı bean'i :

```
public void propertyChange(PropertyChangeEvent event)
{
      if (event.getPropertyName() == "isim")
      {
            newIsim = (String[])event.getNewValue();
            isim = true;
      }
.
.
(diğer indeksli ve tamsayı özellikler için benzer kodlar eklenir)
}
```

Gantt diyagramı bean'i ayrıca bir kaydırma çubuğu içerdiği için, bu kaydırma çubuğunun değerinde meydana gelen değişimleri dinlemek üzere `AdjustmentListener` arayüzü gerçeklenmelidir. Bu kaydırma çubuğunun değeri değiştikçe, diyagramın tekrardan çizilmesi gerektiğinden, bu `AdjustmenListener` arayüzünde yer alan `adjusmentValueChanged` metodu yeniden yazılır ve bu metodun içeriği aşağıda verilmiştir.

```
public void adjustmentValueChanged(AdjustmentEvent e)
{
            value = scroll.getValue();
            repaint();
```

}

Bu bean'de yer alan 'cost' özelliğinin, değerini sadece, kullanıcı girişi bean'inden alması planlandığından, bu özelliğe ait erişim metodlarından sadece belirleyici (setter) metod tanımlanmıştır. Özelliğin değerini alıcı metodun tanımlanmamış olması, ayrıca, bu özelliğe ait bir özellik düzenleyicisinin, bean'in özellik sayfasında yer almaması sonucunu doğurur. Bu da istenen bir sonuçtur, çünkü uygulama geliştirici, bu bean'i düzenlemek üzere bu özelliği belirlememelidir.

Bu bean'de önemli birkaç nokta daha vardır. Bunlardan birisi, kullanıcının girdiği maliyet bilgisinde ve dolayısıyla da kendi içinde yer alan 'cost' değişkeninde bir değişiklik meydana geldiğinde, projeye ayrılan bu yeni maliyet bilgisiyle, her işkalemine ait birim maliyetler üzerinde yapılan bazı hesaplamalar sonunda projenin ne kadar ilerleyeceğini hesaplamakta kullanılmak için yazılan 'setDate' metodudur. Bu metod, yaptığı hesaplar sonucunda, yeni ek maliyetle projenin kaç gün daha ilerleyebileceği sonucunu elde eder ve kaydırma çubuğunun değerini yeni güne ayarlayarak bu yeni tarih ve aralığın çizilmesini sağlar.

Bir diğer nokta da, yer alan getPreferredSize() ve getMinimumSize() metodlarıdır. Bu metodlar, bir Canvas nesnesi kullandığımız için önem taşımaktadır. Bütün kaynaklar, Canvas nesnesinin büyüklüğünün düzgün olarak ayarlanabilmesi için, bu metodların kullanımını tavsiye etmektedirler. Bu metodların kullanımı sonucu, Canvas'ı taşıyıcı nesnenin, onun boyutlarını ayarlarken ve yerleştirirken nasıl davranacağı açıkça belirlenmiş olmaktadır.

4.2.4 Bean'lerin Beanbox Kullanılarak Bağlanmaları

Uygulamamızı oluşturacak üç tane bean, yukarıda açıklanan özellikler ile yazılmışlardır. Bu bölümde, bu bean'lerin uygulama geliştirme aracında nasıl bağlanacakları ve ortaya çıkan uygulamaya ait örnek bir çalışma anlatılacaktır.

Öncelikle, bu bean'lerin nasıl paketlenmeleri gerektiğine bakalım. Her bir bean, bir java kaynak kod dosyası olarak yazılır. Bu dosyalar, .java uzantısına sahiplerdir. Bu kaynak kodlar, JavaSoft tarafından sunulan java derleyicisi (javac) kullanılarak derlendiğinde, bean'ler için Jdk1.1.5 ve üstü tavsiye edilmektedir, .class uzantılı bytekod dosyaları elde edilir. Bu dosyaların, uygulama geliştirme araçları tarafından

görülüp okunabilmeleri için, jar dosyası olarak paketlenmeleri gereklidir. (kodda standart java sınıflarından başka sınıflar da kullanılıyorsa, onlar da pakete eklenmelidir.) Bu paketleme işlemini gerçekleştirmek için gerekli Ms-Dos kod satırı, Gantt diyagramı bean'i için aşağıda verilmiştir.

```
jar cfm GanttChart.jar manifest.tmp GanttChart.class Day.class
```

Bu komut satırı yürütüldüğünde, GanttChart.class ve Day.class (GanttChart kodunda, tarih işlemleri yapmak için kullanılan bir sınıf) sınıfları, manifest.tmp dosyasındaki hangi sınıfın bir JavaBean olduğu bilgisi kullanılarak GanttChart.jar dosyasında paketlenirler. Manifest.tmp dosyasının içeriği aşağıdaki gibidir ve herhangi bir metin düzenleyici kullanılarak oluşturulabilir.

```
Name: GanttChart.class
Java-Bean: True
```

Oluşan jar uzantılı dosya, BeanBox açılırken otomatik olarak yüklenebilmesi için, Bdk\jars dizinine kopyalanır. BeanBox açılırken otomatik olarak yüklenmeyen bir bean, daha sonra 'Load Jar' komutu kullanılarak da yüklenebilir.

Bu uygulama için oluşturulan bean'lerin üçü de, yukarıda verilene benzer şekilde paketlendikten sonra, jar dosyaları, jars dizinine kopyalanmalıdır. Bu aşamadan sonra, artık bir uygulama geliştirici olarak, uygulamamıza uygun olan şekilde bu bean'leri yerel uyarlamamız ve birbirleriyle ilişkilendirmemiz gereklidir. Bu çalışmada gerçeklenmek isteyen uygulamaya yönelik yapılması gerekenler aşağıda verilmiştir.

Öncelikle, beanbox açıldığında karşılaştığımız üç pencereden biri olan en solda yer alan ToolBox penceresinden, uygulamada kullanmak istediğimiz bean'leri seçip, istediğimiz düzende, ortada yer alan BeanBox penceresine yerleştirmeliyiz. Sağda yer alan pencere ise, o an seçili olan bean'in özelliklerine ait düzenleyicileri listeleyen ve bean'in yerel uyarlanmasına imkan sunan Properties penceresidir.

Bu uygulamaya yönelik bean'lerimiz, kullanıcı girişi bean'i CostBox, veritabanı bean'i DBase ve gantt diyagramı bean'i, GanttChart'dır. Bunları sırayla ToolBox'dan seçip, BeanBox'a yerleştirmeliyiz.

Uygulama için, gerekli olan, üç bean arasında üç temel ilişki kurmak ve bazı özellikleri belirlemektir. Öncelikle, CostBox bean'i seçilir. Sırayla Edit | Events | action | actionPerformed seçilir. Bu işlemin sonucunda, CostBox bean'inden çıkan ve fare işaretçisine kadar uzanan bir kırmızı çizgi oluştuğu görülür. Fare, DBase bean'inin üzerine götürülür ve kliklenirse, DBase bean'inin seçilen actionPerformed olayı ile ilişkilendirilebilen metodlarını listeleyen bir pencere görüntülenir. Burada, bizim kodumuzda yer alan dbConnect1 metodu seçilir. Bu işlem sonunda, CostBox içinde 'Enter' tuşuna basıldığında, DBase bean'inin dbConnect1 metodunun çalışması için gerekli ilişkilendirme yapılmış olur. Bu ilişkilendirme, gantt diyagramını çizme ve güncelleme için gerekli başlatma olarak kullanılmaktadır.

İkinci ilişkilendirme, GanttChart bean'inin, DBase bean'inde yer alan bağlı özelliklere kendini dinleyici olarak kaydettirmesi için yapılmalıdır. Aslında bu işlem de, yukarıdaki gibi olay ilişkilendirmesi şeklinde yapılacaktır. Bu sefer, meydana gelen olay, propertyChange, GanttChart bean'inde çağırılacak olan metod da, propertyChange metodudur.

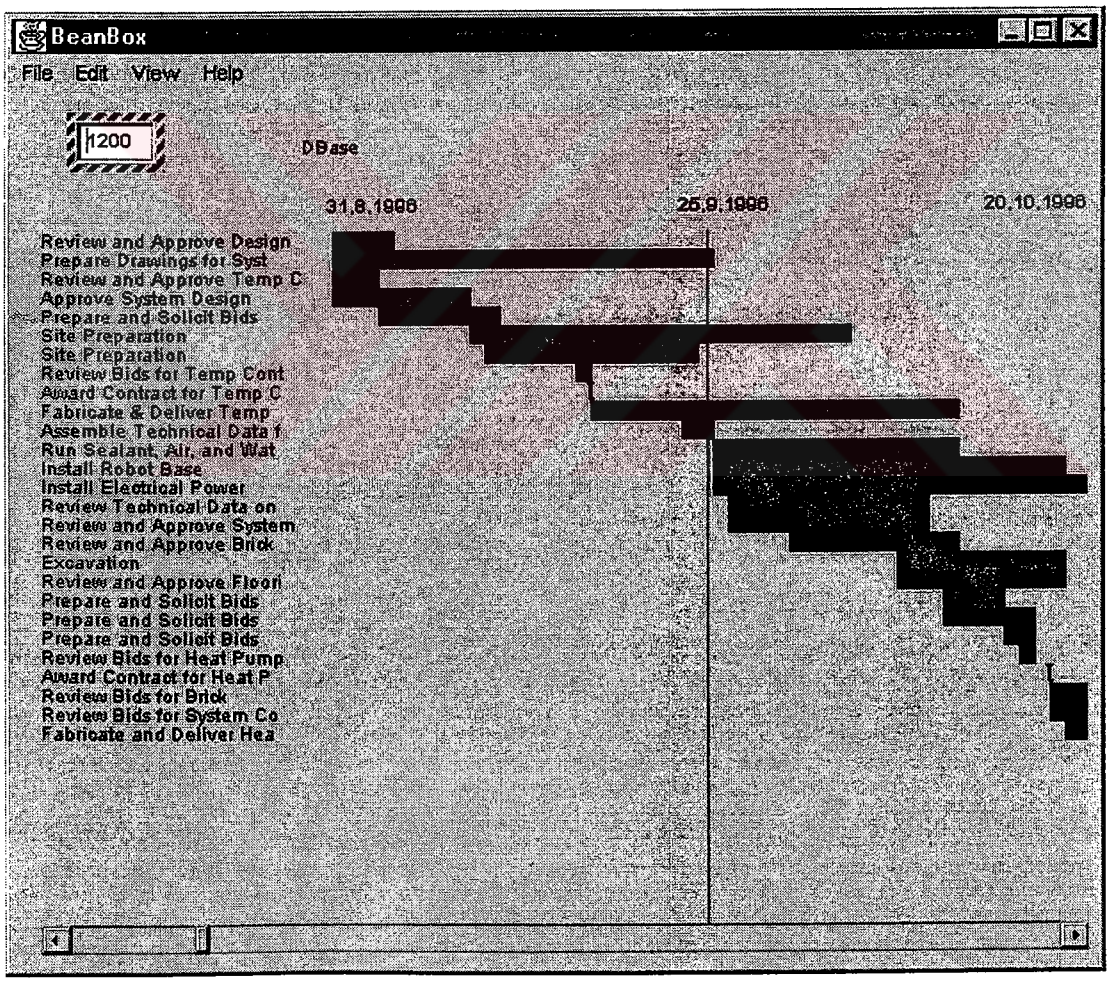
Üçüncü ilişkilendirme ise, CostBox bean'inden alınan 'value' özelliğinin, GanttChart bean'indeki 'cost' özelliğine bağlanması işlemidir. Bunun için, tekrar CostBox bean'i seçilir. Sırayla, Edit | Bind Property seçildiğinde, kaynak özellik seçimi için, o an seçili olan bean'de yer alan mevcut özellikleri listeleyen bir pencere görüntülenir. Bu bean'de yer alan tek özellik olan 'value' seçilir ve mouse, ve dolayısıyla kırmızı çizgi, GanttChart bean'i üzerine götürülüp kliklenir. Bu sefer de, hedef özellik seçimi için, bu bean'de yer alan uygun özellikler listelenir. Bu listede yer alan 'cost' seçilir.

Bu üç işlem sonuçlandığında, bean'ler arasında tanımlanması gerekli tüm ilişkiler belirlenmiş olur. Yalnız bu aşamada, meydana gelen uygulamanın çalışması denenmeden önce, bu çalışmayı doğrudan etkileyen, DBase bean'inin yerel uyarlanması gereklidir. Bu bean için, makinada tanımlanacak Odbc veri kaynağı adı, veritabına bağlantı kurulması sırasında kullanılacak kullanıcı adı ve şifre, sırasıyla, 'dsn', 'passwd' ve 'uname' alanlarına girilir. Bu işlem de yapıldıktan sonra, artık uygulama denenebilir.

İlk çizim için, kullanıcı bean'inin içinden, herhangi bir değer girilmemiş olsa bile 'Enter'a basmak gereklidir. Bu tuş olayının sonucu olarak, veritabanı bean'i

veritabanına bağlanır ve hazırlanan sorguyu yürütür ve elde ettiği sonuçları da uygun dizilere atayarak, özellik değişim olayını ateşler. Bu ateşleme sonucunda, gantt diyagramı bean'i, çizim için kendisine gereken, tüm verileri gerekli nesnelere atayan komutları yürütür ve bu başlatma durumunda, projenin ilk gününden başlamak üzere gantt diyagramını çizer. Uygulama bundan sonra, kullanıcının, kaydırma çubuğunu istediği bir güne götürmesi ve kullanıcı giriş bean'ine, projeye aktarılması düşünülen maliyet miktarını girip 'Enter'a basması ile gantt diyagramı bean'inin, projenin, bu yeni kaynakla, hangi tarihe kadar ilerleyebileceğini hesaplayarak, kendini güncellemesi şeklinde çalışacaktır.

Uygulamanın, tüm bu ilişkiler kurulduktan sonra, beanbox içindeki çalışma anındaki görüntüsü aşağıdadır:



Şekil 4.1 Uygulamanın beanbox içindeki görüntüsü

5. SONUÇLAR VE TARTIŞMA

Bu çalışmada, Proje Yönetim Merkezi için, Bilgi İşlem Merkezi tarafından geliştirilmekte olan otomasyon programının bir bölümü gerçekleştirilmiştir. Geliştirilen Java uygulaması, üç bean'den oluşmaktadır : kullanıcı girişi bean'i, veritabanı erişim bean'i ve gantt diyagramı bean'i. Kullanıcı girişi bean'i, kullanıcıdan bir veri girişi alır. Veritabanı erişim bean'i, yeni veri alındığı zaman veritabanında sorgulama yapar. Gantt diyagramı bean'i ise, veritabanı bean'inden yeni sonuç kümeleri geldiğinde veya kullanıcı girişi bean'inden yeni maliyet bilgisi girildiğinde, yeni verilere göre kendini yeniden çizer. Bu üç bean, bir uygulama geliştirme aracı kullanılarak, yerel uyarlanır, birbiriyle ilişkilendirilir ve ortaya çıkan uygulama, hem geliştirme ortamında, hem de applet haline getirilip, bir inceleyici içinde çalıştırılabilir.

Bu uygulamanın yapısı ve bean'lerin özellikleri, Proje Yönetim Merkezi yöneticileri ile yapılan görüşmeler ve onların istekleri doğrultusunda belirlenmiş ve yapılandırılmıştır. Uygulamanın gelişimi için, gantt diyagramı bean'ine yakınlaştırma ve uzaklaştırma işlevlerinin eklenmesi düşünülebilir. Böylece kullanıcı, proje bilgilerini, daha geniş veya daha dar bir aralıkta görebilir. Ayrıca, Proje Yönetim Merkezi'den gelen yeni talepler doğrultusunda, bean'lerde yapılacak güncellemeler ile uygulamada gerekli değişiklikler sağlanabilir.

KAYNAKLAR

[1] **Hamilton, G.,** 1997. JavaBeans API Tanımlaması. Sun Microsystems.

[2] **Brookshier, D.,** 1997. Java Beans Developer's Reference. New Riders Publishing, Texas.

[3] **Johnson M.**, 1997. A walking tour of JavaBeans, *JavaWorld,* **8.**

[4] **Michael M.,** 1997. Presenting JavaBeans. Sams.net Publishing, Indianapolis.

[5] **Johnson M.**, 1997. Customize Your Java, *JavaWorld,* **9.**

ÖZGEÇMİŞ

E. Esra ATALAY, 1973 yılında İstanbul'da doğdu. 1991 yılında Kadıköy Anadolu Lisesinden ve 1995 yılında İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği Bölümünden mezun oldu. 1995 yılında İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar mühendisliği dalında yüksek lisansa başladı. Aynı tarihte başladığı, İ.T.Ü. Bilgi İşlem Merkezindeki sistem ve ağ yöneticiliği görevini, Ağustos 1998 tarihine kadar devam ettirdi.