JAVA PLATFORMU

Dil ve Platform

Programlama Dili

Java bir programlama dilidir. En basit tanımıyla **programlama dili** (***programming language***), bilgisayarların belli bir işlevi gerçekleştirmesini sağlamak için kullanılan, konuşma diline yakın olan iletişim aracıdır. Programlama dili 'konuşma diline yakındır' ama aynısı değildir. Çünkü konuşma dili bilgisayarın anlayacağı komutlara kolay bir şekilde dönüştürülemez. Bir çok muğlâk, karmaşık hatta edebi ifadelerle doludur. Oysa bilgisayara verilen komutların son derece açık seçik olması gerekir.

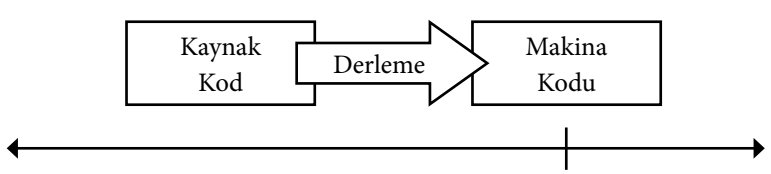
Bilgisayar biliminde 'konuşma dili' yerine '**doğal dil**' (***natural language***) ifadesi kullanılır. Doğal dil, bilim adamlarınca düşünmek ve iletişim kurmak için oluşturulmuş bir dil değildir. İnsan topluluklarının gelişmesiyle birlikte 'doğal' bir biçimde ortaya çıkmıştır. Programlama dilinin doğal dilden farklı olmasının bir başka nedeni de doğal dilin de aslında düşünme diliyle birebir örtüşmemesidir. İnsanlar bir konuda düşünürken belli kavramları ve belli yöntemleri kullanırlar. Birçok insan toplumu kendisine özgü, geçmişi ve edebiyatından etkilenen farklı diller geliştirmiştir. Oysa temelde düşünme, farklı dillerde olsa dahi birbirine yakın şekillerde gerçekleşir. Programlama dili, belli kavramları ve belli işlemleri net bir biçimde ifade edebilme konusunda doğal dile göre çok daha başarılıdır.

Genelde programlama dillerini geliştirenler kullandıkları sözcükleri kendi dillerinden seçerler. Bu yüzden Java dili, kelimelerini **İngilizce**'den almıştır. Ama Java'da kullanılan sözcükler İngilizce kullanımlarından biraz daha farklı, kendine özgü anlamlara sahiptir. Dolayısıyla İngilizce bilmenin Java programlama dilini öğrenmek için fazla bir faydası yoktur. Ancak bir program yazılırken kullanılan isimlerin geleneğe uygun olarak İngilizce seçilmesi nedeniyle, temel İngilizce bilgisi faydalı olabilir. Ayrıca Java hakkında dünyada bulunan belgelerin çoğu, en başta da Java'nın kendi dokümantasyonu İngilizcedir.

Derleme ve Yorumlama

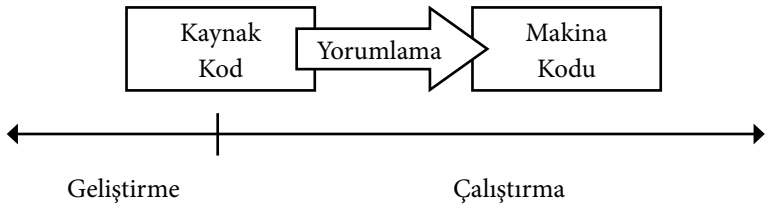
Bilgisayarlarda belli işlevleri gerçekleştirmek üzere kullanılan komutlara '**komut kümesi**' (***instruction set***) denir. Bunlar bir bilgisayar işlemcisinin anladığı sayısal kodlardır. Nasıl yazılırsa yazılsın her program, bir şekilde bilgisayarın anlayabildiği komut kümesindeki komutlara çevrilir. Bilgisayarlar programlama dilini doğrudan anlayamazlar. Dolayısıyla insanların anlayabildiği programlama diline ait sözel komutlar bir biçimde makinenin anladığı sayısal komutlara dönüştürülmelidir.

Bir programlama dilinde yazılmış metne '**kaynak kodu**' (***source code***) denir. Kaynak kodunun yazılması işlemi '**yazılım geliştirme**' (***software development***) veya **yazılım programlama** (***software programming***) olarak adlandırılır. Bir bilgisayarın anlayacağı komut kümesine uygun olarak oluşturulan sayısal kodlara '**makine kodu**' (***machine code***) denir. Makine kodu, işlemcinin anlayabildiği komut kümesindeki komutlardan oluşur. Bir programın makine tarafından anlaşılması için kaynak kodu makine koduna mutlaka dönüştürülmelidir. Bu makine kodlarında belirtilen işler, bilgisayar tarafından yapılır. Buna '**çalıştırma**' (***execution***) denir. Kaynak kodunun makine koduna dönüştürülüp çalıştırılması için iki olasılık bulunmaktadır. Ya bütün kaynak kodu önce tamamen makine koduna çevrilecek ve daha sonra bu makine kodu çalıştırılacak, ya da kaynak kodu programın çalışması esnasında bir yandan makine koduna parça parça çevrilecek, bir yandan da çevrilmiş makine kodu parçaları çalıştırılacaktır.



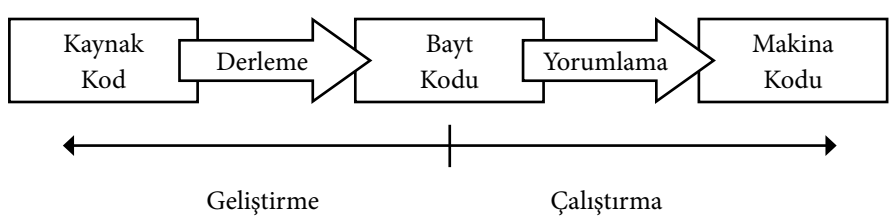
Derlemeli Dillerin Çalışma Biçimi

Tüm kaynak kodunu çalıştırmadan önce tamamen makine koduna çevrilmesi işlemine '**derleme**' (***compilation***) ve bu işlemi yapan programa da '**derleyici**' (***compiler***) denir. Bu şekilde çalışmanın bir avantajı bir de dezavantajı bulunmaktadır. Kaynak kodu bir kere derlendikten sonra bir daha makine koduna çevrilmesi gerekmediğinden program hızlı çalışır. Ancak derleme yoluyla çalışmanın dezavantajı, üretilen makine kodunun sadece belli bir işlemci türünün, belli bir işletim sisteminin anlayacağı komutlardan oluşmasıdır. Programın çalıştırılacağı her işletim sistemi ve her işlemci için programın ayrı ayrı derlenmesi gerekir. Bir işletim sistemi için derlenmiş bir program başka bir işletim sisteminde, hatta bazen bir sonraki veya bir önceki sürümde çalışmayabilir.



Yorumlamalı Dillerin Çalışma Biçimi

Kaynak kodunun bir yandan makine koduna çevrilip bir yandan da çalıştırılmasına '**yorumlama**' (***interpretation***) ve bu işlemi yapan programa da '**yorumlayıcı**' (***interpreter***) denir. Her işletim sistemi (ve işletim sistemi sürümü) için bir yorumlayıcı bulunur. Yorumlayıcı, kaynak kodunu o işletim sistemi için makine koduna dönüştürür ve çalıştırır. Elbette yorumlama ilkesine göre çalışan bir program, bir yandan dönüştürme işlemi yapıldığı için derlenmiş programlara göre daha yavaş çalışır. Ayrıca her program, çalıştırılabilmesi için mutlaka bir yorumlayıcıya ihtiyaç duyar. Bir bilgisayarda yorumlayıcı yoksa kaynak kodu çalıştırılamaz. Oysa derleme yöntemiyle çalışan programlama dillerinde böyle bir sorun olmaz. Programcının makinesinde derleyici olması yeterlidir. Bir makinede üretilen derlenen makine kodu yakın özelliklere sahip olan başka makinelerde sadece kopyalama suretiyle çalıştırılabilir.



Java'nın Çalışma Biçimi

Java platformu **hem derleme hem de yorumlama** yöntemini kullanır. Java programları hemen hemen her makinede, her işletim sisteminde çalıştırılabilirler. O yüzden sadece derleme yöntemi, geleneksel bir biçimde derlenmiş makine kodlarının sadece belli bir makinede çalışabilmesi nedeniyle iş görmez. Buna karşın sadece yorumlama yolu da çok yavaş bir seçenek olduğu için tercih edilmemiştir. Zira tek bir programlama dilindeki komutların çok çeşitli işletim sisteminin makine kodlarına dönüşmesi biraz zaman alır. O yüzden kaynak kodu önce bir kere ara bir dile çevrilir yani derlenir. Sonra ara dildeki kodlar çalışma zamanında çalıştırılır yani yorumlanır. Java'da bu ara dile '**baytkod**' (***bytecode***) adı verilir. baytkod, gerçekte var olmayan, yani 'sanal' olan bir makine için makine kodu olarak düşünülür. Bütün bilgisayarların ortak ve benzer komutlarını çalıştırabildiği var sayılan bu sanal makineye **Java Sanal Makinesi** (**JVM** - ***Java Virtual Machine***) denir. Kaynak kod derlenerek bu makine için yazılmış bayt kodlarına dönüştükten sonra JVM çalışma anında bu kodları gerçek makine kodlarına çevirir ve çalıştırır. Ara dildeki makine kodu gerçek makine kodlarına oldukça yakın olduğu için, kaynak kodunun yorumlanmasına oranla program çok daha hızlı çalışır.

Java'nın, bir ara dile çevrildikten sonra da olsa çalışma zamanında yorumlanı-

yor olması derlemeli dillere göre yavaşlığa, yani düşük performansa yol açar. Ancak Java, programlarını makineye, işletim sistemine ve işlemciye bağımlı olmaktan kurtarır. Makinelerin giderek daha hızlanması ve ucuzlaması, yavaşlık sorununu kritik olmaktan çıkarmıştır. Ayrıca Java'nın çalışma performansı giderek daha da gelişmiştir. Örneğin **JIT** (Tam Zamanında - *Just In Time*) adı verilen bir sistemle, ara dildeki baytkod, çok sık kullanılan kod parçaları için sanal makinenin çalıştığı makinenin koduna bir kez çevrilir. Aynı baytkodun tekrar çağrılması durumunda çevirme yapmaksızın derlenmiş makine kodu doğrudan çağırılır. İşletim sistemlerindeki çeşitlilik, birlikte çalışmak durumunda olan makinelerdeki güç ve özelliklerdeki farklılık ve İnternet'le birlikte birçok makine ve işletim sisteminin birbiriyle bağlantılı olarak çalışmasının gerekliliği, platformdan bağımsızlığı giderek daha da önemli hale getirmiştir. Bu özellik Java platformunun yaygın olarak kullanılır hale gelmesinde etkili olmuştur. Elbette Java'nın geniş bir kitle tarafından benimsenmesinde Java platformunun getirdiği köklü değişikliklerin yanı sıra Java dilinin hem basit hem de güçlü bir dil olmasının da etkisi büyüktür. Ancak Java platformunu kendisinden önceki platformlardan ayıran temel nitelik işlemci ve işletim sisteminden bağımsız çalışması olmuştur.

Java programlarını derlemek için birçok program vardır. Ancak Java platformunun kullanımı zor da olsa temel bir geliştirme aracı bulunmaktadır: *Java Geliştirme Takımı* (**JDK - *Java Development Kit***). Bu sistem bir derleyici program ('**javac**') ve bir yorumlayıcı program ('**java**') içerir. Java programlarını sadece çalıştırmak (yorumlamak) için de Java *Çalışma Zamanı Motoru* (**JRE - *Java Runtime Engine***) adlı bir yazılım bulunur. Birçok internet tarayıcısı Java programlarını bir internet sayfası içerisinde çalıştırmak için bir JVM içerir.

Java'nın kaynak kodları '**.java**' uzantılı metin dosyalarında bulunur. Derleyici bu dosyaları derler ve bayt kodların bulunduğu '**.class**' uzantılı dosyaları üretir. İkili kodlar, yani sadece 1 ve 0'lardan oluşan değerler içeren dosyalar, bir programın çalışması için yeterli olur. Genellikle, tek bir birimde bulunmaları ve daha az yer kaplamaları için bir programa ait '.class' dosyaları sıkıştırılmış halde '**.jar**' uzantılı dosyalarda tutulur. **JAR** kısaltması Java Arşivi demektir ve temelde sıkıştırma standardı ZIP biçimine dayanır.

Temel Özellikler

Nesneye Yönelik Programlama

Java nesneye yönelik bir dildir. En basit tanımıyla '**nesne**' (***object***), veri ve işlev içeren bir bütündür. Günlük dildeki 'şey' sözcüğünün yazılımdaki karşılığı nesnedir. Gerçek dünyadaki her 'şey' programlamada bir nesne ile simgelenebilir. Örnek nesne olarak dikdörtgenler prizması ele alınabilir. Dikdörtgenler prizması 'genişlik', 'yükseklik' ve 'derinlik' verilerine sahiptir. Java'da bunlara '**özellik**' (***property***) denir. Dikdörtgenler prizması nesnesinde 'hacim hesaplama' ve 'dış yüzey alanı hesaplama' şeklinde iki işlev bulunur. Java'da işlevlere '**yöntem**' (***method***) denir.

Programlamada iki temel unsur vardır: **veri** (***data***) ve **işlev** (***function***). Java'da veriler nesnelerin özelliklerinde, işlevler de nesnelerin yöntemlerinde tanımlanır. Örnekte 'genişlik', 'yükseklik' ve 'derinlik' birer veridir. Buna karşın 'hacim hesaplama' ve 'alan hesaplama' ise birer işlevdir. Nesneler verileri ve işlevleri bir arada tutan yapılardır. Hem özellik hem de işlev bir nesneye **üye** (***member***) durumundadır. Başka bir deyişle özellik bir nesnenin veri içeren, yöntem ise işlev içeren üyesidir.

Gerçek dünyada da her nesneyle ilgili belli veriler ve işlevler vardır. Java'da nesneler kullanarak gerçek dünyadaki soyut veya somut nesneleri tanımlayabiliriz. Nesneler kullanarak program yazma tarzı, bir yazılım sisteminin gerçek dünyadaki sisteme yakın bir yapıda kurulabilmesine olanak tanır. Nesnelerle program yazma tekniğine '**nesneye yönelik programlama'** (***object-oriented programming***) denir. Java'da hemen her şey nesnedir ve nesneleri bir kez kavrayan kişi Java'nın temel mantığını kavramış olur. Java'da yapılan her şey hazır bulunan nesnelerin nasıl kullanıldığını öğrenmek ve gerektiğinde çeşitli nesneler yapmaktan ibarettir. Nesne yapmadan, sadece hazır nesneleri kullanarak program yapmak da mümkündür. Buna '**nesne tabanlı programlama**' (***object-based programming***) denir. Nesneye yönelik programlama, nesnelerle birlikte 'kalıtım' (*inheritance*), 'çok biçimlilik' (*polymorphism*) ve 'kaplama' (*encapsulation*), gibi temel kavramlar içerir.

Kalıtım (*Inheritance*)

Genel anlamıyla **kalıtım** (***inheritance***), bir nesnenin başka bir nesnenin özelliklerini alabilmesidir. Örnek olarak bilgisayar bir elektronik aygıttır. Her elektronik aygıt elektrikle çalışır. O zaman bilgisayar da elektrikle çalışıyor demektir. Nesneye yönelik programlama diliyle konuşursak: '*bilgisayar elektrikle çalışma niteliğini elektronik aygıttan kalıtır*'.

Çok Biçimlilik (*Polymorphism*)

Nesneye yönelik programlamanın bir başka kavramı olan **çok biçimlilik (*polymorphism*)**, birbirine yakın nesnelerin işlevlerinin aslında farklı çalışmalarına rağmen aynı şekilde kullanılmasıdır. Örneğin televizyon da, bilgisayar da birer elektronik cihazdır. İkisinin de güç düğmesi vardır, basarsak ikisi de açılır. Aslında iki aygıt açılırken çok farklı işlemler yapar ama biz bununla ilgilenmeyiz. İki farklı aygıtı aynı şekilde açarız. Nesneye yönelik programlama diliyle konuşursak, '*aç yöntemi televizyon ve bilgisayarda çok biçimlidir*'.

Kaplama (*Encapsulation*)

Tanım olarak **kaplama** (***encapsulation***), bir nesnenin belli üyelerinin, başka nesneler tarafından kullanımının kısıtlanabilmesidir. Böylece bir nesne başka bir nesnenin verilerini ve işleyişini bozamaz. Bir özellik veya yöntem 'özel' (*private*), 'kamusal' (*public*) ve 'korumalı' (*protected*) olarak belirlenebilir. Bir üye için **özel** (***private***) denirse sadece o nesne tarafından kullanılabilir, başka nesneler okuyamaz ve değiştiremez. Üyeye **kamusal** (***public***) denirse o özellik herkes tarafından okunur ve değiştirilir. Eğer üye **korumalı** (***protected***) olursa bir nesnenin diğer bir nesnenin özelliklerini okuması ancak o nesneden türetilmesi durumunda mümkündür anlamına gelir.

Basitlik ve Gelişmişlik

Java, 'güçlü' bir dil olmasına rağmen diğer bazı 'güçlü' dillere göre çok basit bir dildir. Elbette Java'dan çok daha basit, çabuk öğrenilen ve bir işin çok çabuk bitirilebildiği birçok programlama dili bulunmaktadır. En başta da **JavaScript**. Bu dil, sınırlı özelliklere sahiptir ama kolay öğrenilip kolay kullanılabilir. Java'ya yakın bir söz dizimi vardır ve Java ile iletişim kurma yeteneğine sahiptir. Ancak ne JavaScript ne de diğer kolay diller gelişmiş özellikler içermez. Programcının ileri düzeyde, uzun vadeli bir yazılım yapısı kurması için gerekli temel araçlardan yoksundur. Öte yandan, yazılım dünyasında birçok gelişmiş dil bulunmaktadır. Ancak bunların da öğrenilmesi ve uygulanması Java kadar kolay değildir. Örneğin **C/C++**. Basit söz dizimine az sayıda anahtar sözcüğe sahip bu dil zamanla ilerlemiş ve nesneye yönelik programlama gibi birçok gelişmiş özelliğe sahip hale gelmiştir. Donanıma bağlı yazılımlarda ve yüksek performans isteyen işlemlerde C/C++ çok başarılıdır. Java'nın kendisi C/C++'la yazılmaktadır. Java, performans ve donanıma bağımlı işlemler için bu dilde yapılmış kütüphaneleri kullanmaktadır. Bir C/C++ kodu Java'da doğrudan çağrılamaz. **JNI** adı verilen Java'nın '**yerli kod'** (***native code***) yani işletim sisteminin makine kodunu çağırabilmesini sağlayan altyapı bulunmaktadır. Programcı C/C++ programlarını çağırmak için JNI sistemine uygun C/C++

kodu yazmak durumundadır. C/C++ nin öğrenilmesi ve kullanılması oldukça zordur. Programcının çok dikkatli olmaması durumunda çok ciddi hatalar ortaya çıkabilir. Bu yüzden de hızlı geliştirme isteyen, hatayı az hoş gören ve çok sayıda programın birlikte çalıştığı ortamlarda kullanımı giderek azalmıştır. Programlama dillerinin birçoğu 'güçlü ama zor' ile 'kolay ama zayıf' sıfatlarından birine uyar. Java hem güçlü hem de basit olma özelliğine sahip olma iddiasında olan bir dildir.

Anahtar Sözcük (*Keyword*)

Bir programlama dili '**anahtar sözcük**' (***keyword***) adı verilen çeşitli sözcüklerden oluşur. Bunlar sayesinde programcı, yapılmasını istediği tanımları ve işlemleri belirtir. Anahtar sözcük ifadesi yerine '**ayrılmış sözcük**' (***reserved word***) ifadesi de kullanılır. Çünkü programlama dili belli sözcükleri kendine ayırır, programcı herhangi bir şeyin ismi olarak bu sözcükleri kullanamaz. Ancak Java'da ayrılmış sözcükler arasında yer alan ama anahtar sözcük olmayan sözcükler de bulunur. Bunun dışında, aslında anahtar sözcük olmayıp, 'doğru', 'yanlış' veya 'boş' gibi ***yalın (literal)*** özel değerler bildiren ve isim olarak kullanılamayan ifadeler de bulunmaktadır.

Java, söz dizimi açısından model aldığı C dili gibi az anahtar sözcüğe sahiptir. Bunun nedeni, eklenen her anahtar sözcüğün, dili eskisinden daha karmaşık hale getirmesidir. Java dili sadece nesneleri tanımlamaya ve kullanmaya yarayan anahtar sözcüklerden oluşur. Bazı dillerde belli bir anahtar sözcük tarafından yapılan işlem Java'da bir nesne tarafından gerçekleştirilir. Anahtar sözcükler sadece bir nesne tarafından gerçekleştirilmesi mümkün olmayan işlevler için kullanılır. Bu durumda kod biraz daha uzun yazılmakla birlikte, öğrenilecek anahtar sözcük sayısının azlığı, yeni öğrenenlerin dile daha çabuk hâkim olmalarını sağlamaktadır.

Java'da kullanılan anahtar sözcükler (*keywords*) ve kullanılmayan ancak ayrılmış sözcükler (*reserved words*) ve anahtar sözcük gibi görünen ancak aslında yalın *(literal)* değer olan sözcükler aşağıda listelenmektedir.

İlk sürümden itibaren bulunan anahtar sözcükler	**abstract continue for new switch default package synchronized boolean do if private this break double implements protected throw byte else import public throws case instanceof return transient catch extends int short try char final interface static void class finally long volatile float native super while**
Sonraki sürümlerde eklenen anahtar sözcükler	**strictfp** (1.2 sürümünde eklendi) **assert** (1.4 sürümünde eklendi) **enum** (1.5 sürümünde eklendi)
Anahtar olmayan ama ayrılmış olan sözcükler	**goto** **const**
Anahtar sözcükmüş gibi görünen ancak yalın değer olan sözcükler	**true** **false** **null**

Kütüphaneler (*API*)

Her dil; programcıların kullanımına hazır, önceden tanımlı birçok nesne veya işlev içerir. Programcı bir işi gerçekleştirirken bu hazır kütüphaneleri kullanarak hem o işi daha çabuk bitirmiş olur hem de kullanılan kütüphaneler daha önce çok kez kullanılıp test edilmiş olduğu için hız ve hatasızlık bakımından kazanç elde edilmiş olur. Belli konularda birbiriyle yakın işlevler gören nesnelerin toplandığı kümeye '**kütüphane**' (***library***) veya **API** (Uygulama Programlama Arayüzü – *Application Programming Interface*) denir. Her kütüphane belli bir konuda işlev gören çok sayıda nesneden oluşur. Java platformuyla birlikte gelen kütüphanelere '**standart kütüphane**' denir. Henüz standart olmamış, ama Java'yı oluşturan kurumlarca üzerinde çalışılmış, kullanıcının gerektiğinde kurup kullanabildiği kütüphanelere '**seçmeli kütüphane**' denir. Bunların dışında standart olarak kabul görmemiş ancak birçok kişi ve kuruluşun üzerinde çalıştığı yaygın olarak kullanılan kütüphaneler de vardır. Java, her konuda 'özgür' veya 'bedava' kütüphanelerin çok sayıda bulunduğu bir dildir. Ancak Java platformunun standart kütüphaneleri de oldukça zengindir ve giderek daha da zenginleşmektedir. Her yeni Java sürümü birçok kütüphaneyi ve nesneyi bünyesine katmaktadır.

Java'da birbiriyle ilgili sınıflardan oluşan bütüne **paket** (***package***) denir. Kütüphaneler bir kaç paketin birleşiminden oluşur. Paket adları *xxx.yyy.zzz* biçiminde nokta ile ayrılmış isimlerden oluşur. Java'nın standart paketleri **java.** veya **javax.** adı ile başlar. Kurumlara ait paketler de org veya com adıyla başlar. Örneğin *org.apache.xxx* veya *com.sun.xxx* şeklinde bir isim 'xxx' paketinin sırasıyla Apache örgütü ve Sun şirketine ait olması durumunda aldığı isimdir.

Java'da çok sayıda paket bulunmakla birlikte bunlardan bazıları çok kullanılan sınıfları içermektedir. Bunlardan **java.lang** Java programlama dilinin kendisiyle doğrudan ilgili temel sınıfları içermektedir. Bu pakete ek olarak **java.util** çok kullanılan, genellikle veri yapılarıyla ilgili bir çok yararlı sınıf içermektedir. Dosya işlemleri gibi giriş çıkış sınıfları **java.io** paketinde bulunmaktadır. Ağ programlamayla ilgili temel sınıfların bulunduğu paket de **java.net** kütüphanesidir. Temel grafikli arayüz geliştirme ve işletim sisteminin grafik ortamıyla ilgili sınıflar içeren AWT kütüphanesi **java.awt** ve alt paketlerinde bulunmaktadır. Daha zengin bir arayüz geliştirme için gerekli kütüphane **javax.swing** ve alt paketlerinden oluşur.

Java'da bir kütüphane ve içerisindeki sınıflar hakkında **belgeleme** (***documentation***) yapmak için **JavaDoc** denilen bir sistem bulunur. Buna göre her sınıf, özellik ve yöntem hakkında bilgi, kaynak kodunun içerisine yazılır. Java geliştirme ortamında bulunan **javadoc** programı bütün kaynak kodlarını tarar, içerisindeki belgeleme özelliklerini ayırır ve çeşitli belgeler üretir. Belgelemenin ayrı yapılmayıp doğrudan kaynak koduna gömülü olmasının yararı, belgelemenin çok kolay olmasıdır. Programcının yapacağı şey nesnelerin içerisine birkaç satır yazı yazmaktan ibarettir. Bu şekilde kaynak kodundaki değişiklikler belgelemeye otomatik olarak yansır. Aksi takdirde kaynak kodu değişince belgelemenin neresi nasıl değişmeli diye bulmak zor olurdu. En çok kullanılan belgeleme türü HTML biçimidir. Programcı bir kütüphane hakkındaki bilgiye, JavaDoc aracıyla oluşturulmuş HTML belgelerinde gezinti yaparak ulaşabilir. Java'nın standart kütüphanelerinden üretilmiş *Java Standart API Documentation* biçiminde adlandırılan belgeleme bulunmaktadır. Java'da bulunan çok sayıda sınıfın ezberlenmesi gerekmez. Temel mantığın anlaşılması ve kütüphanelerin genel olarak işlevlerinin bilinmesi yeterlidir. Gerektiğinde belgelemeye bakılarak yeterli bilgi alınabilir.

Java'da nesnelere ve çeşitli nesne öğelerine isimler verilirken belli bir isimlendirme geleneği (*naming convention)* standardına uyulur. Bu sayede kaynak kodunu okuyan veya belgelemeye bakan bir kişi nesne hakkında bir fikir sahibi olur. Derleme bakımından anahtar sözcük olmaması koşuluyla herkes her istediği ismi vermekte özgürdür. Ancak isimlendirmede standartlara uymak

kodun okunaklılığını büyük ölçüde artırır. Standart isimlendirmeye alışan kişi başkalarının yazdıklarını da daha rahat okumaya başlar. Bu, özellikle bilinmeyen bir konuda araştırma yapıldığında bulunan örnekleri anlamakta oldukça yararlı olur. Geliştiriciler arasında bilgi paylaşımı yazılım geliştirmenin ayrılmaz bir parçasıdır. Dolayısıyla isimlendirme geleneği bir 'özgürlük kısıtlaması' veye 'katı kuralcılık' değil, teknolojide uyum ve iletişim sağlamanın en önemli unsuru olan standartlaşmanın doğal bir sonucudur.

Java'nın kütüphanelerinde **belirtim** - **gerçekleştirme** (***specification*** - ***implementation***) ayrımı bulunmaktadır. Burada belirtim, kütüphanenin tanımlanması demektir. Sadece geliştiricinin bu kütüphaneden bekleyebileceği işlevler, sadece onun kullanabileceği sınıflar ve arayüzlerden oluşur. Başka bir deyişle asıl işlevler tanımlanır, işlevlerin nasıl gerçekleştirildiği belirtilmez. Gerçekleştirim de bir belirtimin gerçekleştirme biçimidir. Bir belirtim birden fazla kurum tarafından gerçekleştirilebilir. Gerçekleştirimlerden bir tanesi daha hızlı çalışması için yapılırken bir diğeri daha az bellek harcamak için tasarlanabilir. Ancak geliştirici bunlarla ilgilenmez, gerçekleştirimi hiç görmeden kullanır. Java'nın hemen hemen bütün kütüphaneleri böyledir. Kullanıcıyı ilgilendirmeyen sınıflar kesinlikle ortada görünmez. Belirtim bir çok firma, kuruluş ve kişinin bir araya gelip belirlediği bir yapıdır. Sonra herkes kendi gerçekleştirimini yapabilir. Ancak bir gerçekleştirimden diğerine geçildiğinde geliştirici için hiçbir sorun ortaya çıkmaz, kodu tekrar yazmak zorunda kalmaz. Java'nın kütüphaneleri bir çok kurum tarafından gerçekleştirilmiştir. Ama hepsinin kullanımı aynıdır. Geliştirici her biri için ayrı bir kod yazmaz. Ancak bu belirtim-gerçekleştirim ayrımının bir olumsuz yanı, geliştiricinin yazdığı kodun biraz daha karmaşık olmasıdır.

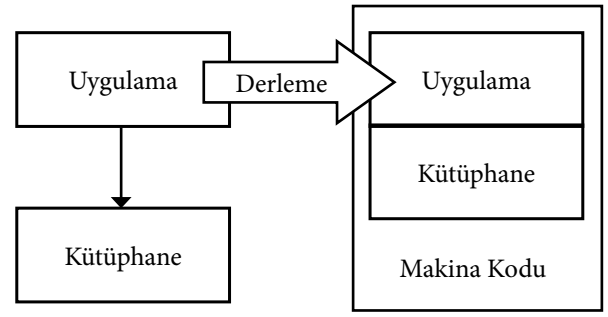
Geriye Uyumluluk

Java'nın standart kütüphaneleri '**geriye uyumlu**' (***backward-compatible***)dur. Bunun anlamı programcının kullandığı nesnelerin yeni bir Java sürümü çıktığında da çalışmasıdır. Sürüm uyumsuzluğu diye bir sorun ortaya çıkmaz. Bu sayede Java'nın ilk sürümü üzerinde çalışan program son sürümde de aynen çalışır. Elbette hatalı veya kullanılması zararlı nesneler veya nesnelerin bazı üyeleri olabilir. Bu durumda bunları Java '**eskimiş**' (***deprecated***) diye işaretler. Eskimiş nesne ve üyeler eskisi gibi çalışmaya devam ederler ama yeni sürümlerde onların kullanılmaması gerektiğini ve yerlerine neyin nasıl kullanılması gerektiğini belirten uyarılar gelir. Elbette eskiden yazılmış programlar için bu uyarı anlamsız olacağından onlar ilk yazıldıkları zaman olduğu gibi eskimiş nesneleri kullanmaya devam ederler. Java'nın geriye uyumluluk özelliği, yeni sürümlere geçişi daha az sorunlu hale getirmektedir. Aksi takdirde ya yeni

sürümlere geçip eski sürümelere göre yazılmış programları elden geçirmek veya kısmen yeniden yazmak zorunda kalınırdı ya da yeni sürüme geçmekten vazgeçilirdi.

Dinamiklik

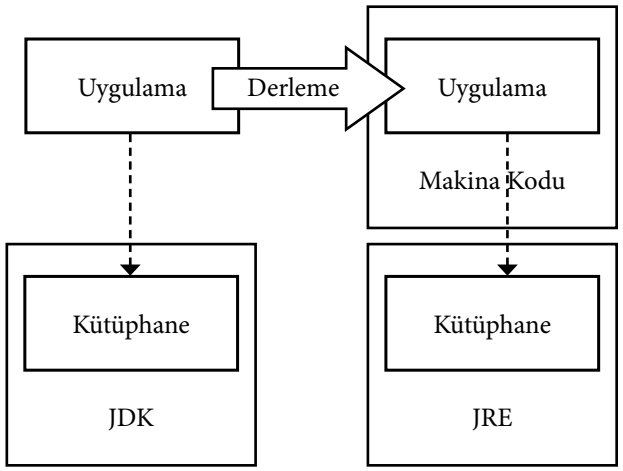
Java '**geç bağlama**' (***late-binding***) ilkesine göre çalışır. Yazılımda geç bağlama ve '**erken bağlama**' (***early-binding***) bir nesnenin kullandığı nesneyle olan bağlantısının ne zaman yapıldığıyla ilgili kavramlardır. Java öncesi dillerde bir program bir kütüphanedeki herhangi bir nesneyi kullandığı zaman o kütüphanenin bütün kodu kullanan programa eklenir. Örneğin, *bir* nesneden



Erken Bağlama

oluşan bir program *yüz* nesneden oluşan kütüphaneyi kullandığında *yüz bir* nesnelik bir ikili kod oluşur. Oysa Java'daki geç bağlama özelliği bir nesnenin bir kütüphaneden bir nesne kullanması durumunda derleme yaparken sadece kullanan nesneye, kullanılan nesnenin adını ve erişilen kısımlarının çağrılışlarını yazar. Kullanan nesnenin kullanılan nesneye gerçekten erişmesi çalışma zamanında gerçekleştirilir. Yani kullananın kullanılana bağlanması 'erken' (derleme zamanında) değil 'geç' (çalışma zamanında) olur. Program sadece programcının yazdığı nesnelerden oluşur, kullandığı hiçbir nesne programın içinde yer almadığı için programlar çok fazla yer kaplamaz ve bir yerden diğerine taşınmaları kolay olur. Ayrıca kütüphanelerdeki olası değişikliklerden, değişiklik çağırış şeklini etkilemediği sürece program etkilenmez.

Bir makinede Java platformu yüklüyse Java ile gelen bütün kütüphaneler de yüklü demektir. Bu durumda programcı kendi makinesinde kendi Java kurulumundaki kütüphanelerdeki nesnelere göre derleme yapar. Ancak programın çalışacağı makineye sadece program gider. Çünkü kullanılan kütüphaneler o makinede de bulunmaktadır. Gereksiz yere kütüphanelerin programla birlikte taşınması söz konusu olmaz.



Geç Bağlama

Çok Akışlı (*Multi-Threaded*) Programlama

Java '**çok akışlı**' (***multi-threaded***) bir platformdur. Çok akışlılık, bir programda aynı anda birden fazla işlem yapılabilmesidir. Bu sayede program bir şeyi beklerken arada başka işler de yapabilir. Bir akışta yavaş bir işlem yürütülürken başka bir akışta hızlı bir işlem başlatılıp bitirilebilir. Bir programın aynı anda birden fazla kullanıcı tarafından kullanılması veya bir programa birden fazla kaynaktan veri gelip gitmesi durumunda da çok akışlılık gereklidir. Çok akışlılık Java'da bir kütüphane olarak değil, platformun temel özelliği olarak bulunmaktadır. Programda ayrı bir akış oluşturulmasa dahi programın kendisi '**ana akış**' (***main thread***) adı verilen akışta çalışır.

Java'da çok akışlı programlama üzerine bir de **eşakışlı programlama (*concurrent programming*)** kütüphanesi bulunmaktadır. Bu kütüphane çok akışlı programlama yapmak için belli bir çerçeve sağlar ve çok karşılaşılan sorunlara standart çözümler üretir. Elbette programcı eşakışlı programlamada bulunan nesneleri çok akışlı programlamayla kendi yapabilir. Ancak yapılması gerekenler eşakışlı programlamada bulunan standart nesnelerde bulunuyorsa bu bir zaman kaybı olur ve hata yapma riski doğar.

Platform Bağımsızlığı

Java '**platformdan bağımsız**' (***platform independent***) veya başka bir deyişle '**mimarilere yansız**' (***architecture neutral*)** bir yazılım platformudur. Herhangi bir bilgisayar işlemcisi üzerinde, güçsüz bir makineden en güçlü sunucu makinelerine kadar geniş bir yelpazedeki donanımlarla, hemen hemen bütün işletim sistemleriyle ve sürümleriyle çalışabilir. Java öncesi dillerde bütün sistemlerde aynı *kaynak kodunun* kullanılması büyük ölçüde başarılabilmiştir.

Ancak *ikili kodun* bütün sistemlerde çalışmasını ilk sağlayan Java olmuştur. Platform'dan bağımsızlık programcıyı her sistem için farklı sürümler yapmaktan kurtarır. Bir programa sahip olanlar gerektiğinde işletim sistemini değiştirebilirler ve kullandıkları Java programlarını yeni işletim sistemlerinde aynen veya çok az değişikliklerle kullanmaya devam ederler.

Java'nın platformdan bağımsız olmasın ilginç bir avantajı da, bazı kurum ve kuruluşların belli bir işletim sistemi üzerinde geliştirdiği yazılım kütüphanelerinin başka işletim sistemleri için de kullanılabilmesidir. Java öncesi diller için bu söz konusu değildir. Bunlarda her işletim sistemi kendi kütüphanelerini geliştirir. Başka bir mimariye aktarmak için kodu '**taşıma**' (***porting***), yani diğer işletim sistemine uyarlama gereklidir. Ama Java kendinden '**taşınabilir**' (***portable***) bir platformdur.

Dağıtık Programlama

Java, dağıtık programlamayı destekler. En basit tanımıyla **dağıtık programlama (*distributed programming*)** bir programın birden fazla makineye dağıtılmış bir biçimde çalışabilmesidir. Programın bir parçası bir makinede çalışırken aynı anda diğer bir parça başka bir makinede çalışabilir ve bunlar çalışma zamanında birbirleriyle iletişim kurabilirler. Dağıtık programlama, yapılacak programların tek bir makineye sığmaması durumunda yazılımın **ölçeklenebilir** (***scalable***) olmasını yani daha büyük kapasitelerde çalışabilir hale gelebilmesini sağlar. Ayrıca farklı yazılım sistemlerinin birbiriyle çalışma zamanında **bütünleşik** (***integrated***) çalışmasını sağlar.

RMI

Java'da dağıtık programlama için birkaç seçenek vardır. Bir tanesi **RMI** (**Uzak Yöntem Yakarımı** – ***Remote Method Invocation**) protokolüdür.* Bir makinede çalışan Java nesnesinin başka bir makinedeki nesneyi sanki kendi makinesinde çalışıyormuş gibi kullanabilmesidir. Bu sayede bir yazılım sistemini oluşturan nesneler farklı makinelere dağıtılır. Yazılım bir makinede değil bütün bir ağda bulunur. RMI sadece Java platformuna özgü bir yapıdır. Yani birbiriyle konuşan nesneler hep Java nesneleri olmalıdır.

CORBA

Dağıtık programlama için bir başka seçenek **CORBA** (**Ortak Nesne İstek Aracısı Mimarisi** – ***Common Object Broker Architecture**) protokolüdür.* CORBA protokolünün özelliği dağıtık programlamayı farklı diller arasında yapabilmesidir. Bir ağa yayılmış nesnelerden biri Java'da diğeri C++'da yapılmış olabilir. IDL (***Interface Definition Language*** - **Arayüz Tanımlama Dili**) adı verilen, çeşit-

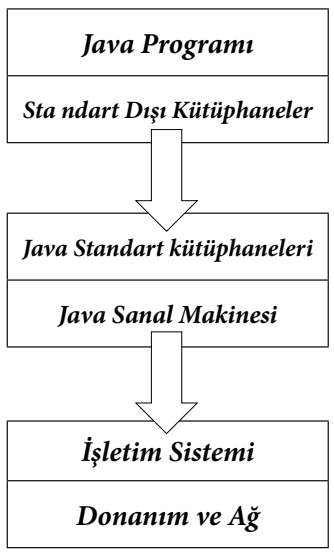
li dillerin desteklediği, dillerden bağımsız bir arayüz (arabirim), yani ortak bir yapı tanımlama dili bulunur. Her dil, kendi nesnelerini IDL kullanarak tanımlar. CORBA altyapısı da bu dile göre nesnelerin birbiriyle konuşmasını sağlar.

Web Servisleri

Dağıtık programlama için bir seçenek de **Web Servisleri** (***Web Services***) seçeneğidir. Web Servisleri, bir programın başka bir programla iletişimin Web üzerinden XML aracılığıyla kurulduğu sistemdir. İnternet'in en çok kullanılan protokolü (iletişim kuralları bütünü) **HTTP** üzerinden çalışır. **XML**, genel amaçlı bir tanımlama dili olduğu için dağıtık programlamada da kullanılması düşünülmüştür. Farklı makineler aralarında XML diliyle tanımlanan **SOAP** protokolüyle konuşurlar. Web Servisleri **WSDL** diye bir XML diliyle tanımlanırlar ve tanıtılırlar. Bir web servisini bulmak için de **UDDI** adlı bir başka XML dili kullanılır.

Seçeneklerin Karşılaştırılması

RMI, sadece Java'nın desteklediği bir protokoldür. CORBA ise Java, C++ gibi nesneye yönelik diller için çalışan, sadece belli platformların desteklediği bir protokoldür. Web Servisleri hemen hemen bütün platformlarca desteklenir. Örneğin Microsoft .NET platformu da Web Servislerini destekler. Bir Java nesnesi ile bir C# nesnesiyle yazılmış bir web servisini çağırabilir. Ancak HTTP üzerinden metin olarak XML kullanarak iletişim kurduğu için ikili kodlarla çalışan diğer iki seçeneğe göre daha yavaştır.



Java Çalışma Zamanı Katmanları

Sağlamlık

Java platformu 'sağlam' bir platformdur. Java programları, kullanıcılar veya farklı herhangi bir etken tarafından yapılan hatalar karşısında kolay kolay çökmez. Java platformu sağlamlık için bazı önlemler almıştır. Java dilinin **derleme zamanında hata yakalama** özelliği oldukça gelişmiştir. Yani programcı hata içeren bir kodu çoğu kez derleyemez bile. Derleme aşamasında anlaşılması mümkün olmayan hatalar da **çalışma zamanı hata yakalama** özelliği sayesinde program çalışırken yakalanır. Hata oluşsa da program çökmez. Başka programları, hele işletim sistemini tümden çökertmek gibi bir durum hemen hemen hiç oluşmaz.

İstisna (*Exception*)

Hata oluşması durumunda Java dilinin '**istisna yönetimi**' (***exception handling***) mekanizması devreye girer. Yazılımda **istisna** (***exception***) terimi normalde karşılaşılması beklenmeyen durumları, örneğin hataları belirtir. İstisna yönetimi, programcının oluşan hatayı anlayıp önlem almasına olanak tanır. Bir işlem yapılırken hata oluşması durumunda yapılacaklar belirtilir. Hata oluştuğunda programcının belirttiği işlemler sırasıyla gerçekleştirilir. Programcı hiç bir önlem almasa da Java yakaladığı hatayı ekrana (veya bir dosyaya) ayrıntılı bir biçimde yazar.

Belleğe Erişim Sınırlaması

Java'da **belleğe erişim** (***memory access***) engellenir. Bu sayede bir programın başka bir programın verilerine erişmesi ve onu bozması mümkün olmaz. Java'da bellek gerektiren işlemler belleğe erişim yapılmadan yapılır. Bazı programlama dillerinde 'işaretçi' diye bir kavram vardır. Bir **işaretçi** (***pointer***), bellekte belli bir nesnenin yerini yani adresini belirtir, oraya 'işaret' eder. Programcı bu işaretçiyi kullanarak veriyi bulunduğu yerden kullanabilir. Kullanıldığı her yere taşımak zorunda kalmaz. Ancak işaretçi, hafızadaki belli bir adresi gösterdiği için, adres bilgisinin yanlışlıkla veya kasıtlı olarak başka bir adrese yönlendirilmesi veya bir adresteki veriye yanlış bir fonksiyonun erişmesi sorun yaratır. Programın hatta işletim sisteminin çökmesine yol açabilir. Java bellekte belli bir yer belirten 'işaretçi' yerine 'başvuru' kavramını kullanır. Bir **başvuru** (***reference***), bellekte belli bir nesneyi belirtir fakat bu yer belirtme nesnenin bellekteki gerçek konumuyla olmaz. Bellekteki gerçek adres Java platformu tarafından gizli olarak tutulur. Programcı sadece elindeki bu başvuruyla çalışır. Bir başvuru sadece o veri veya nesne için geçerlidir. Kazayla veya kasten hafızadaki başka bir nesne için kullanılamaz. Başvurular kullanılarak bellek işlemleri, belleğe doğrudan erişilmeden gerçekleştirilmiş olur.

Bu şekilde üretilen yazılım da daha sağlam olur. Ancak bunun bedeli olarak Java programları gerçekten belli bir bellek alanına değil de gerçek bir bellek adresine gereksinim duyulan işlerde, yani donanıma bağlı işlerde doğrudan kullanılamaz. Zaten Java platformunun hedefi uygulama programlamadır, donanım veya sistem programlama değil. Donanımla veya sistemle ilgili işlevler genellikle C/C++'la yapılır ve Java'dan kullanılması gerekiyorsa **JNI** adı verilen, Java'nın C++ kodlarını çağırmasını sağlayan sistemle dolaylı olarak kullanılır.

Otomatik Çöp Toplama (*Garbage Collection*)

Java'da otomatik '**çöp toplama**' (***garbage collection***) sistemi vardır. Çöp toplama, kullanılmayan nesnelerin bellekten otomatik olarak boşaltılması demektir. Java öncesi dillerde programcı kullandığı belleği kendisi boşaltmak zorundadır. Boşaltmazsa '**bellek sızıntısı**' (***memory leak***) yani kullanılmak için ayrılmış ama artık kullanılmayan bellek alanları oluşur. Bellek sızıntıları, programın uzun süre çalışması durumunda belleğin giderek tamamını kaplar, sonunda da kullanılan asıl veriler için yer kalmaz. Birçok programın bir süre çalıştıktan sonra yavaşlaması ve en sonunda da çökmesinin nedeni budur. Java'da kullanılmayan nesnelerin yer kaplaması çöp toplama ile engellenir. Java sanal makinesi bir nesnenin kullanımlarını izler ve o nesne hiçbir yerden kullanılmamaya başlanınca o nesneyi bellekten boşaltır. Bu otomatik yapıldığı için bellek sızıntısı oluşmaz.

Güvenlik

Java platformu İnternet ve ağ dili olarak tasarlandığı için güvenliğe çok önem vermiştir. Aksi takdirde sağladığı geniş olanaklar Java'nın kötü amaçlarla kullanılmasına yol açabilirdi.

Java ikili kodları (baytkodları), Java Sanal Makinesi (JVM) tarafından yüklenirken '**bayt kod doğrulayıcı**' (***bytecode verifier***) adı verilen bir sistem tarafından kontrolden geçirilir. Bayt kod doğrulayıcı yüklenen her nesneyi bazı kontrollerden sonra çalıştırır. Nesnenin, erişme hakkına sahip olduğu nesnelerin, erişme hakkına sahip olduğu üyelerini çağırıp çağırmadığı denetlenir. Nesnenin bellekte taşmaya veya sızıntıya yol açacak işlemler yapıp yapmadığına bakılır. Java programlarına **virüs bulaşmaz** çünkü virüslü dosya, baytkod doğrulayıcı tarafından fark edilir ve çalıştırılmaz. Ayrıca Java programlarının belleğe erişemiyor olmaları da güvenlik açıklarının en önemli nedeninin ortadan kalkmasına yol açar. Bir programın başka bir programın şifre gibi güvenlikle ilgili verilerine erişmesi Java'da olanaksızdır.

Güvenlik Yöneticisi (*Security Manager*)

Standart güvenlik özellikleri dışında Java platformunda '**güvenlik yöneticisi**' (***security manager***) kavramı vardır. Bir programın çalıştığı ortamdaki bütün hareketler güvenlik yöneticisi tarafından kontrol edilir. Çalıştığı makinenin diskine erişim, bağlı bulunan aygıtlara erişim, ağ üzerinden başka herhangi bir makineye erişim hep bu güvenlik yöneticisi tarafından kontrol edilir. Bazı ortamlar her türlü erişime izin verilmeyi beklerken bazı ortamlar da bazı erişimlerin sınırlanmasını gerektirir. Örneğin bir tarayıcı içerisinde çalışan Java uygulamacığı yani '*applet*', özel olarak izin verilmedikçe çalıştığı makinede hiç bir yere erişemez. Kendi geldiği makine dışında hiçbir makineye bağlanamaz. Erişim hakkının applet tarafından alınması ancak kullanıcının bilerek yetki vermesiyle gerçekleşebilir.

Güvenlik Kütüphaneleri

Java'nın güvenli yapısının dışında oldukça zengin **güvenlik** (***security***) ve **şifreleme** ***(cryptography)*** kütüphaneleri vardır. Güvenlik kütüphaneleriyle bir programın belli kaynaklara erişimin sınırlandırılması ve yetkilendirme gibi standart güvenlik işlevleri platformun bir parçası olarak bulunmaktadır. Şifreleme kütüphanesi ise, bir verinin bir yerden başka bir yere aktarılırken şifreli tutulması, gerektiğinde sadece yetkili kişilerce bu şifrenin çözülmesi gibi konuları içerir.

Uygulama Alanları

Java platformunun birçok aygıtta çalışan sürümü bulunmaktadır. Masaüstü bilgisayarlar, sunucular, cep telefonları, el bilgisayarları, akıllı kartlar ve televizyonlar bunlar arasında sayılabilir. Bilgisayarlar içerisinde de çeşitli yazılım geliştirme alanlarında Java ile programlama yapılabilmektedir. Java'da masaüstü makineler için **Standart Sürüm (*Standard Edition*)**, sunucular ve kurumlar için **Kurumsal Sürüm (*Enterprise Edition*)**, taşınabilir aygıtlar için ise **Taşınabilir Sürüm (*Mobile Edition*)** bulunmaktadır**.**

Konsol Uygulaması

Masaüstü veya sunucu bilgisayarlarda grafikli pencere veya web sayfası gibi ara yüz ortamlarına gerek duymadan çalışan Java uygulamaları, **konsol** (***console***) uygulamalarıdır. Konsol, pencere veya düğme gibi grafiksel olmayıp sadece harf ve sayılarla kullanıcıyla iletişim kurulmasını sağlayan işletim sistemi ortamıdır. Kullanıcı ile çok fazla etkileşim gerektirmeyen veya belli işlerin toplu yapıldığı programlar konsol programları için en uygun alanlardır. Ayrıca konsol programları Java'yı öğrenme aşamasında, ara yüz nesnelerinin karmaşık yapılarına girmeden temel nesnelerle çalışmak için de kullanılır. Bir başka kullanım alanı da yapılan nesnelerin doğru çalışıp çalışmadığının anlaşılması için bir test uygulaması yazılmasıdır. Her nesne veya birkaç nesneden oluşan grup, küçük bir konsol uygulamasıyla test edilir ve sorunsuz çalıştığı anlaşıldıktan sonra asıl programda kullanılmaya başlanabilir. Konsol uygulamaları **DOS** veya **Shell** gibi işletim sistemlerinin metin tabanlı ortamlarında kullanılır. Bütün metin tabanlı ortamlarda çalışan programlar gibi, konsol programı adının yazılmasıyla çalıştırılabilir. Programa parametre aktarmak mümkündür. Konsoldan veri girişi yapılabilir ve konsola yazı şeklinde bir çıktı verilebilir.

Masaüstü Uygulaması

Masaüstü uygulamaları, bir pencere içerisinde çalışan görsel bileşenler içeren uygulamalardır. Kullanıcı; düğme, menü gibi bileşenlerde fareyle tıklayarak yapmak istediklerini yapar. Masaüstü uygulamalarının kullanılmadan önce bir makineye kurulmaları gerekir. Standart olarak Java'da iki tip ara yüz sistemi bulunmaktadır: AWT ve Swing. **AWT** adlı kütüphane, işletim sisteminin ara yüz bileşenlerinin kullanılmasıyla çalışır. Elbette program hangi işletim sisteminde çalışıyorsa çalışma zamanında onun bileşenleri kullanılır. Örneğin 'düğme' (*button*) bileşeni işletim sisteminin kendi düğmesidir, Java ortamı sadece Java uygulaması ile işletim sistemi arasında arabuluculuk yapar. AWT,

bütün işletim sistemlerinde çalışmak zorunda olduğu için bunların hepsinde bulunan ortak bileşenleri kullanır. Bu ortak bileşenler de bütün işletim sistemlerinde bulunan ortak özellikleriyle kullanılır. Bir işletim sisteminde olan bir bileşen diğerinde yoksa veya bir özellik diğerinde bulunmuyorsa AWT sisteminde de yoktur. Bu durum AWT altyapısının biraz daha basit ve sınırlı bir sistem olmasına yol açar.

İkinci arayüz sistemi **Swing** kütüphanesidir. Swing'de işletim sisteminin görsel bileşenleri kullanılmayıp özel bileşenler yapılmıştır. Elbette işletim sisteminin grafik altyapısını kullanılır. Swing, AWT alyapısı üzerine kurulmuştur ancak AWT kütüphanesindeki görsel bileşenler yerine '**J**' harfiyle başlayan kendine özgü bileşenler barındırır. Swing kütüphanesi zengindir ancak işletim sisteminin bileşenlerini kullanmayıp kendi bileşenlerini kendi yaptığı için AWT kütüphanesine göre daha yavaştır. Düşük yapılandırmalı makinelerde bazı performans problemlerine yol açabilir. Ancak zamanla makinelerin ve grafik kartlarının kapasiteleri arttıkça Swing uygulamalarıyla işletim sisteminin kendi yerli uygulamaları arasındaki performans farkı giderek kapanmıştır. Swing uygulamaları, üzerinde çalıştıkları işletim sisteminin arayüzüne görünüş olarak benzeyebilir. Swing kütüphanesinde hiçbir işletim sistemi arayüzüne benzemeyen, tamamen Java tarafından yapılmış arayüz görünüşleri de bulunmaktadır. Hatta programcılar kendi ara yüz tercihlerinden bir görünüş oluşturup programlarının buna uygun olarak görünmesini sağlayabilirler.

Tarayıcıda Uygucuk (Applet)

Bir tarayıcı (*browser*) içerisinde bir web sayfasında sıradan bir görsel bileşen gibi çalışan Java nesnelerine **applet**, yani '**uygucuk**' denir. Bunlar web sayfalarına programlama becerisi kazandırmış olurlar. Masaüstü ve konsol uygulamalarının aksine bir makineye kurulmaları gerekmez. Bir web sayfasının içerisine yerleştirilir ve sayfa indirilirken tarayıcı tarafından otomatik olarak yüklenir ve çalıştırılır. Applet, çabuk kullanıcı etkileşimi isteyen oyun ve çevrimiçi uygulamalar tarafından tercih edilen bir teknolojidir. Çünkü web sayfalarının sunucu tarafına veri aktarmaları zaman kaybına yol açar. Uygucuklar bir tarayıcı içerisinde güvenli bir şekilde çalıştırılırlar. İndirildiği makinede diske veya bağlı herhangi bir aygıta erişim yapamazlar. İndirildikleri makine dışında hiçbir makineye bağlanamazlar.

Web Start Uygulaması

Masaüstü programlarıyla uygucuk (applet) teknolojisinin üstün yanlarını birleştiren yapı **Web Start** teknolojisidir. Bir masaüstü uygulamasının bütün

özelliklerine ve haklarına sahip olabilir. Ancak kurulumları bir applet gibi kolay bir şekilde web üzerinden olur. Uygulama ilk kez çalışırken web üzerinden indirilir. İkinci kez çalışırken yerel makineden çabucak yüklenebilir. Uygulamanın yeni sürümü çıktığında otomatik olarak makinedeki eski sürüm üzerine güncelleme yapılabilir. Web Start uygulamaları bir kez yüklendikten sonra istenirse çevrimdışı yani İnternet'e bağlı olmadan da çalıştırılabilir.

Web Uygulaması

Web uygulaması **istemci** (***client***) tarafında değil **sunucu** (***server***) tarafında çalışan, istemci ile bir tarayıcı üzerinden HTML diliyle iletişim kuran bir uygulama türüdür. Java da **Servlet** adlı nesneler web uygulaması olarak görev yaparlar. Servlet uygulamaları bir **web sunucusu** (***web server***) veya **uygulama sunucusu** (***application server***) içerisinde çalışır. Bir sunucuya bağlanan çok sayıda kullanıcının yönetilmesi uygulama sunucusu tarafından otomatik olarak yapılır. Servletler bütün web uygulamalarında olduğu gibi istemciden parametre ve girdileri almak ve istemciye HTML veya başka bir biçimde çıktı vermek suretiyle çalışırlar. Servlet uygulamaları birer nesnedir ve Java dilinin standart çalışma tekniğine uygun olarak çalışır.

Java'da servlet teknolojisi üzerine kurulmuş **JSP** teknolojisi bulunmaktadır. Servlet uygulamaları istemci tarafına bütün HTML belgesini üretip göndermek zorundadır. Yapılan programın az, üretilen HTML belgesinin çok olması durumunda işler biraz zorlaşır. JSP ise Java kodlarını HTML içerisine gömme yöntemiyle çalışır. JSP belgeleri de aslında birer servlete dönüştürülür. İlk çağrıldıkları zaman otomatik olarak Servlet sınıfına dönüştürülür ve sonra da çalıştırılır. Bazı web uygulamaları; arayüzün ağırlıklı olduğu durumlarda JSP, çalışma mantığının ağırlıklı olduğu durumlarda Servlet kullanılan karma bir yapı da sergileyebilirler. Servlet uygulamaları ve JSP sayfaları birbirlerini çağırabilir, birbirlerine veri aktarımı yapabilirler.

Ağ Uygulaması

Java'da **sunucu** (***server***) veya **istemci** (***client***) yazmak mümkündür. Web, e-posta, oyun, sohbet sunucuları yazıldığı gibi, Java servlet'lerini çalıştırmak için kullanılan Java uygulama sunucuları da bulunmaktadır. Sunuculara bağlanan istemci programları da yazılabilmektedir. Ağ programlama, İnternet'in temel protokolü **IP** ***(Internet Protocol)*** üzerinden yapılır. Bu altyapı üzerinde bilgisayarlar arasında veri iletişimi kurulur. İletişimin iki biçimi vardır: TCP ve UDP. **TCP** bir makineden bir makineye veriyi gönderir ve gidip gitmediğini kontrol eder. **UDP** ise veriyi gönderir ve gidip gitmediğiyle ilgilenmez.

UDP protokolünde verinin güvenliği garanti edilemez ancak bir seferde birçok makineye aynı anda veri gönderilebilir. Bütün İnternet protokolleri bu yapılar üzerine kurulur. HTTP web sisteminin ana protokolüdür; SMTP, POP3, IMAP e-posta protokolleridir; FTP dosya transferi protokolüdür. IRC de UDP protokolüne dayanan bir sohbet protokolüdür.

Java, İnternet ve network programlamayı destekler. Bunun için '**priz**' (***socket***) adı verilen altyapı iki makine arasında iletişim kurmakta kullanılır. Bunun dışında, network programlama yapmadan makineler arası veri aktarımını gerçekleştirebilmek için HTTP, FTP gibi protokolleri desteklemektedir. HTTP protokolünün güvenli biçimi olan HTTPS de Java tarafından desteklenen protokoller arasındadır. E-posta göndermek ve almak gibi işlevler için Java'da standart bir e-posta kütüphanesi bulunmaktadır.

Kurumsal Sürüm (Enterprise Edition)

Java'da **Kurumsal Sürüm** (***Enterprise Edition***) denilen, bir kurumun içindeki çok sayıda bilgisayarın karşılıklı iletişim halinde çalışması için alt yapı oluşturan bir sürümü bulunmaktadır. Servlet ve JSP dâhil Java'daki birçok teknolojiyi içermektedir. Java'nın temel dil özellikleri geçerli olmakla birlikte standart Java'nın ötesinde dağıtık, kalıcı, güvenli, ölçeklenebilir, taşınabilir vs. birçok özelliği olan nesnelerden oluşan bir altyapı sunar.

Mobil Uygulama

Java'nın bir başka uygulama alanı da cep telefonu ve el bilgisayarı gibi taşınabilir aygıtlardır. Burada kullanılan temel nesne MIDlet sınıfıdır. **Taşınabilir Sürüm** (***Mobile Edition***), standart Java'ya göre biraz daha sınırlı özellikler içerir. Çünkü cep telefonu gibi aygıtlar bilgisayar kadar güçlü özellikler içermez. Ancak Java dili temelde aynıdır, sadece kullanılan nesneler ve kütüphaneler değişir. Java'nın temel nesnelerinin birçoğu 'Taşınabilir Sürüm'de de bulunmakta, taşınabilir aygıtlar giderek geliştikçe daha çok standart Java kütüphanesi taşınabilir sürüme de dahil edilmektedir. Bir taşınabilir uygulama sunucuda bir yere yerleştirilir ve gerektiğinde Java taşınabilir platformu tarafından cep telefonuna otomatik olarak yüklenir.

Java'nın Diğer Kullanım Alanları

Java yukarıda sayılan alanlar dışındaki alanlarda da kullanılır. Örneğin **TV** üzerinde program yapmak için **Xlet** nesneleri yapılabilir. Akıllı kartların birçoğunda **JavaCard** platformu yüklüdür. Akıllı kartlar üzerine veri yazan veya okuyan, onların üzerinde çeşitli şekillerde çalışan uygulamalar Java ile yazılabilir.

Temel Araçlar

JVM ve JRE

Java uygulamalarını çalıştırmak için bir **JVM** (**Java Sanal Makinesi**- ***Java Virtuval Machine***) olması gereklidir. JVM, işletim sistemine gömülü olabilir, tarayıcı gibi bir programın içerisinde bulunabilir veya bir motor olarak makineye kurulduktan sonra Java uygulamalarının çalıştırılmasını sağlayabilir. İzni olan her şirket kendisine bir JVM yapabilir. Piyasada çeşitli işletim sistemleri için çok sayıda JVM bulunmaktadır. Bazı işletim sistemi üreticileri, kendileri için JVM de üretmişlerdir. Bazılarında Java işletim sitemiyle birlikte gelir, bazılarında ayrıca kurmak gerekir.

Java uygulamalarını çalıştırmak için kullanılan JVM gerçekleştirimlerinden en çok bilineni **JRE** (**Java Çalışma Zamanı Motoru** - ***Java Runtime Engine***) programıdır. JRE, bir JVM ve yardımcı bir takım programlar içerir. Örneğin JRE programına Web Start altyapısı da dâhil edilmiştir. JRE Java Plugin olarak da adlandırılır. Çünkü JRE bir tarayıcının Java desteğine sahip hale gelmesini sağlar. JRE ücretsiz birçok yerden indirilebilen bir yazılımdır.

JDK ve Java Geliştirme Ortamları

Java kaynak kodlarını derlemek için kullanılan araç JDK (Java Geliştirme Takımı - *Java Development Kit) ortamıdır.* Derleyici içerdiği gibi bir JVM (JRE) de içerir. Bunun dışında çeşitli geliştirme araçları da JDK ile birlikte gelir. JDK, Java ile geliştirme yapmak için gerekli her aracı içerir. Ancak bunlar kullanımı kolay, görsel araçlar değildir. Bir çok konsol uygulamasından oluşan bu araçları yeni öğrenenler kullanmakta zorlanabilirler.

Java ile geliştirme yapmak için çok sayıda **geliştirme ortamı** ve **kod düzenleyici** bulunmaktadır. Birçoğu ücretsizdir veya sınırlı ücretsiz sürümlere sahiptir ve İnternet'ten indirilebilir. Kullanımı kolay menüler ve araç çubuklarından oluşan bu ortamlar içlerinde JDK içerirler. Temelde yaptıkları, JDK takımının daha kolay kullanılmasını sağlamak ve kod geliştirmek için programcılara çeşitli kolaylıklar sunmaktır.

Java Geliştirme Ortamları Hakkında

Her geliştirme ortamı kendine özgü özelliklere sahiptir. Ancak pek çoğu birçok ortak nokta içerir ve temel çalışma ilkeleri birbirine benzer. Burada geliştirme ortamları hakkında genel bilgiler verilmiştir.

JDK

Bazı ortamlar **JDK** takımını da birlikte kurarlar. Bazılarında JDK takımını ayrı olarak kurmak ve ortama JDK takımını tanıtmak gerekir. Genelde tanıtma JDK programının kurulu olduğu klasörü (*'java home' değerini)* göz atıp seçmekten ibarettir.

Proje

Bazı ortamlar doğrudan bir Java dosyası açıp düzenlenmesini sağlarken bazıları bütün Java kaynak kodlarının **proje** (***project***) adı verilen birime bağlı olmasını isterler. Proje, belli bir konuda yazılan bütün nesne ve diğer dosyalardan oluşan bütündür. Bazı ortamlarda öncelikle bir proje açılmalı ondan sonra Java kaynak kodları yazılmalıdır. Bazıları açılışta örnek bir proje açarak programcıyı bu külfetten kurtarır. Geliştirme ortamlarından bazıları projeleri de gruplamak için *solution*, *workspace* gibi birimler tanımlanmasına olanak tanımaktadırlar.

Dosya

Java sınıfları 'java' uzantılı metin dosyalarında durur. Yeni bir sınıf oluşturmak için '**yeni**' (***new***) benzeri bir menü maddesi içerirler. Bazıları yeni sınıf, yeni dosya derken bazıları da baştan oluşturulacak sınıfın *applet*, *servlet*, *console application*, *Swing application* şeklinde türünün belirtilebilmesini de destekler. Eğer bunlardan biri yapılmak isteniyorsa yeni sınıf veya yeni dosya yerine bu seçenekler seçilebilir. Seçenekler arasında genelde '***class***' (**sınıf**) seçeneği bulunur ve bu sıradan bir sınıf açar. Bu sınıfta bazı şeyler programcıya kolaylık olsun diye hazır yazılmış olabilir.

Derleme

Bütün geliştirme ortamları derlemeye yarayan menü maddelerine (ve araç çubuğu düğmelerine) sahiptir. Bazılarında bu araçlar '***compile***' *(***derle***)* bazılarında '***make***' (**yap**) ve '***build***' (**yapılandır**) gibi adlar alırlar. İşlevleri, kaynak kodu ikili koda dönüştürüp bir programı çalışabilir hale getirmektir.

Çalıştırma

Geliştirme ortamları, derlenmiş ikili kod çalıştırma özelliğini de içerirler. Bazıları '***run***' (**koştur**) bazıları da '***execute***' (**yürüt**) adını alırlar. Bazı ortamlar 'çalıştır' deyince aynı zamanda derlemeyi de yapar. Elbette bir Java sınıfının çalıştırılabilir olması için hem derlenebilir hem de çalıştırılabilir bir biçimde yazılmış olması gerekir.

JDK Kurulum ve Kullanılması

Neden JDK?

JDK takımını kullanmak bir geliştirme ortamı kullanmaya oranla çok daha zordur. O yüzden yeni başlayanların bir geliştirme ortamı kullanarak uygulama geliştirmeye başlaması çok daha uygun olur. Yine de JDK ortamının kullanılmasının faydalı olabildiği durumlar vardır. Bir makinede geliştirme ortamı kurmak, kapasite düşüklüğü gibi sebeplerle mümkün değilse JDK bir seçenek oluşturabilir. Bazı işletim sistemlerinde geliştirme ortamı bulmak bazen zor olabilir veya bazı işletim sistemi kullanıcıları arayüzlü çalışmak yerine komut satırından çalışmayı tercih edebilirler. Birçok komut satırı aracı içeren JDK bu durumda iyi bir seçenek oluşturabilir.

Geliştirme ortamları kolay olmalarına rağmen çok sayıda işlem için birçok pencere açıp kapama, birçok tıklama yapmayı zorunlu kılmaktadır. Oysa bazı araçlar, komut satırından birçok işlemi JDK ortamını kullanarak tek komutla yapabilir. Hatta bunun için altyapı sağlayan, JDK üzerine yapılandırılmış çeşitli programlar da mevcuttur. Örneğin '***ant***'. Komut satırı araçlarıyla çok sayıda klasör açma, derleme, arşivleme, dosya kopyalama, belge oluşturma gibi işlevleri tek bir komutla yapmak olanaklıdır.

JDK ortamını kullanmanın bir yararı da, Java'nın son sürümlerini takip edebilme kolaylığıdır. Yeni bir Java sürümü çıktığında öncelikle JDK çıkar. Aslında Java'nın yeni sürümünün çıkması demek yeni bir JDK ortamının çıkması demektir. Geliştirme araçlarının yeni Java sürümünü destekleyen sürümlerinin çıkması biraz zaman alır. Bazı şirketler son Java sürümünü destekleyen geliştirme araçlarını parayla, eski sürümleri destekleyenleri bedavaya verir. Java'nın yeni sürümleri daha çıkmadan henüz beta aşamasındayken de JDK ortamına erişmek mümkün olabilir.

Geliştirme ortamları da aslında JDK'yı kullanır. Ön tarafta kullanıcıyla iletişim kuran görsel araçlar, arka tarafta derleme ve çalıştırma işlemlerini JDK komut satırı araçlarını kullanarak gerçekleştirir. O yüzden JDK'yı öğrenmek geliştirme ortamlarının temel özelliklerini anlamak açısından faydalı olabilir.

Bütün bu sayılan yararlarına rağmen JDK ortamıyla çalışmak zordur. Burada, yine de kurmak isteyenler veya kurmasa da en azından geliştirme ortamlarının temel mantığını öğrenmek isteyenler için bilgi verilmiştir.

JDK Araçları

JDK ile birlikte Java ile geliştirme yapabilmek için gerekli çok sayıda araç gelir. Bunlar JDK'nın kurulduğu klasörde **bin** klasöründe bulunan '**exe**' uzantılı dosyalarda bulunurlar. Bunlardan en temel olanları Java sınıflarını, yani ikili kodlarını yorumlamaya, yani çalıştırmaya yarayan **java.exe**, Java kaynak kodlarını derlemeye yarayan **javac.exe**, Java sınıflarını ve onlar tarafından kullanılan dosyaları tek bir dosya biçiminde arşiv oluşturmaya veya arşivi açmaya yarayan **jar.exe**, Java kodları hakkında belgeleme oluşturmaya yarayan **javadoc.exe** gibi araçlardır.

JDK Kurulumu

Aşağıda JDK kurulumu Windows işletim sistemine göre adım adım anlatılacaktır. Adımlar içerisinde DOS (komut istemi) ortamına geçmek gerekir. Diğer işletim sistemlerinde yapılacaklar bunlara paraleldir.

JDK Takımının İndirilmesi

JDK bir diskten alınır veya İnternet'ten indirilir ve diskin bir klasörüne kopyalanır. (JDK ortamının indirilebileceği yerler bir arama motorunda '*JDK Download*' anahtar sözcükleriyle bulunabilir.) Örneğin;

```
http://www.xxx.com/java/download.html
```

gibi bir sayfadaki 'download' şeklinde bir bağlantıdan gelen "jdk-xxx-yyy.exe" gibi bir dosya "*c:\jdownload*" şeklinde bir dizine indirilir.

JDK Kurulması

Diskte bulunan '*exe*' uzantılı dosya çift tıklamak suretiyle veya komut satırında adı yazılarak çalıştırılır. Bu program çoğunlukla '*next*' düğmeleriyle, bazen makinenin yeniden başlatılmasını gerekli kılacak şekilde ilerler. Kurulum için önerilen klasör '*c:\jdk*' şeklinde değiştirilse faydalı olur. Zira kurmak için programın yaptığı seçim her zaman uygun olmayabilir. Kurulumu test etmek için komut satırında **-version**

```
c:\jdk\bin\java -version
```

ifadesi yazılabilir. Burada java.exe aracına sürümün basılması için version parametresi verilmektedir. Eğer bu ifade girildikten sonra ekranda 1.6.XXX gibi bir değer görülüyorsa ve gerçekten de kurulan sürüm bu basılan değerse kurulum başarılı olmuş demektir.

Kaynak Dosyası Oluşturma

Çalışma Dizini Yaratma

Diskte Java kaynak kodlarının ve ikili dosyaların bulunacağı dizin (adı '*c:\jwork*' olsun) yaratılır.

Bir Java Dosyası Oluşturma

Açılan çalışma dizininin içerisinde örnek olarak adı '*MyClass*' uzantısı '***java***' olan bir metin dosyası oluşturulabilir. Bunun için komut satırında çalışma dizinindeyken '*notepad MyClass.java*' yeterli olabilir. Oluşturulan bu dosyanın içine,

```
public class MyClass{
  public static void main(String[] args){
   System.out.println("Tamam!");
  }
}
```

şeklinde metin yazılır. Bu bir deneme nesnesidir ve görevi, ekrana 'Tamam!' şeklinde bir yazı yazmaktır.

Derleme Aracı

Java kaynak kodlarını derlemek için JDK ile birlikte gelen **javac** aracı kullanılabilir. Bu araç JDK'nın kurulduğu yerde bulunan **bin** klasörü içindeki **javac.exe** adlı programa karşılık gelir. Bu araç komut satırı ortamında gerekli değerlerin parametre olarak aktarılmasıyla kullanılır.

Kaynak Kodu Derleme

Oluşan **.java** uzantılı dosyayı komut satırında (DOS ortamında) ***javac*** aracı kullanılmak suretiyle,

```
c:\jdk\bin\javac MyClass.java
```

ifadesi yazılarak derlenebilir. Bunun sonucunda aynı dizinde '*MyClass*' adında '***class***' uzantılı ikinci bir dosya oluşturulmuş olmalıdır.

Kaynak ve Sınıf Dizinleri

Derleme işleminde eğer üretilen ikili dosyaların başka bir dizine konması isteniyorsa derleme aracı **-d** parametresiye bir dizin belirtilerek çağrılır :

```
c:\jdk\bin\javac -d c:\myclasses MyClass.java
```


Kaynak dosyalarının bulunduğu konumu belirtmek için de **-sourcepath** (**kaynak yolu**) parametresi kullanılabilir :

```
  c:\jdk\bin\javac -sourcepath c:\mysources
MyClass.java
```

Yorumlama Aracı

Java ikili dosyalarını çalıştırmak için, JDK ile birlikte gelen veya JRE içinde bulundan **java** aracı kullanılabilir. Bu araç JDK veya JRE'nin kurulduğu yerde bulunan **bin** klasöründeki **java.exe** programına karşılık gelir. Bu araç komut satırında gerekli değerlerin parametre olarak aktarılmasıyla kullanılır.

İkili Kodu Çalıştırma

Oluşan '*class*' uzantılı dosyayı çalıştırmak için sınıf dosyasının bulunduğu klasörde (örnekte *c:\jwork* klasöründe)

```
c:\jdk\bin\java MyClass
```

ifadesi yazılır. Ekrana 'Tamam!' şeklinde bir ifade çıkarsa JDK başarılı bir şekilde kurulmuş ve çalıştırılmış demektir. Artık Java çalışmak için gereken her şey makinede bulunmaktadır.

Özellik Değeri Aktarma

Bazı durumlarda java derleyicisine belli özelliklerin aktarılması gerekebilir. Bu değerler java.exe tarafından doğrudan kullanılmayabilir. Sanal makine içinde çalışan sınıflar veya modüller bu değerleri kullanıyor olabilir. Özelliklerin kullanımları ve adları çok çeşitli olabileceği için belli bir parametre adı yerine **-D** ile başlayan özellik adlarıyla parametre aktarımı yapılır. Örneğin 'abc' özelliğinin değeri 'xyz' ise çalıştırma

```
java -Dabc=xyz
```

biçiminde gerçekleşir. Örneğin Java'da OpenGL yönteminin DirectX olması için *-Dj3d.rend=d3d* gibi bir ifade yazılır.

Ortam Değişkenleri

JDK veya JRE araçlarının çalışması için gerekli çok sayıda ortak değer bulunmaktadır. Bunlar her çalıştırmada araçlara tek tek verilmesi yerine işletim sistemlerinin **ortam değişkeni** (***environment variable***) denilen değerlerinde tutulur. Bunlar araca ilişkin programa parametre aktarılmadan, program tarafından ortamdan alınarak kullanılır.

Java Home ('Java Evi')

Java'nın, yani JDK'nın bulunduğu klasöre ***java home*** (**java evi**) denir. Java evi işletim sistemlerinde bulunan **ortam değişkeni** olarak ***JAVA_HOME*** olarak tanımlanabilir. Bu değerin atanması, komut satırında,

```
set JAVA_HOME="c:\jdk"
```

biçiminde bir ifade yazılarak yapılabilir. Eğer Java evi ortama tanıtılırsa, hem java derleyicisi, hem yorumlayıcısı hem de java programlarının çalışması kolaylaşır. Programcı her seferinde Java'nın bulunduğu klasöre referans vermek zorunda kalmaz. Yani komut satırında c:\jdk\java demek yerine java demesi yeterli olur. Java ortamının kurulduğu yeri ve Java programlarının kullandığı standart Java kütüphanelerini bulabilmesi için 'java.exe' yorumlayıcı programının bu değişkeni bilmesi gerekir.

Classpath ('Sınıfyolu')

Bir nesnenin, kullandığı diğer nesneleri veya nesnelerden oluşan kütüphaneleri bulmak için bakılacak klasörler veya sıkıştırılmış dosyalar listesi '***classpath***' *(***sınıfyolu***)* değeridir. Komut satırında kullanılan '*java*' komutuna bir programı çalıştırmasını söylerken nesnelerin derlenmiş ikili dosyalarını nereden bulacağını da söylemek gerekir. Bu yüzden java yorumlayıcısına '**-classpath**' veya kısa hali **-cp** gibi bir parametre ile bu bilgi aktarılır. Örneğin çalıştırma işlemi,

```
c:\jdk\bin\java -classpath "c:\jwork" MyClass
```

biçiminde yapılabilir. Paralel bir biçimde derleyiciye gerekli diğer sınıfların bulunduğu yeri bildirmek için,

```
c:\jdk\bin\javac -classpath "c:\orherclasses"
MyClass.java
```

biçiminde çağrım yapılabilir. Sınıfyolunu her çalıştırmada araçlara parametre olarak aktarmamak için '**CLASSPATH**' adlı ortam değişkeni kullanılabilir. Buna göre;

```
set CLASSPATH="c:\xxx;%CLASSPATH%"
```

ifadesi *java.exe* çalıştırılmadan önce bir kez girilir. Sınıfyolu kullanarak yapılan işlem Java ortamına bir sınıfın kullandığı diğer sınıfların nereden bulunabileceğini söylemektir.