Program Nedir ?

Sözlükte program kelimesi, belirli şartlara ve düzene göre yapılması öngörülen işlemlerin bütünü ve aynı zamanda izlence olarak tanımlanmıştır. Yazılımcılara göre Program, bilgisayara bir işlemi yaptırmak için yazılan komutlar dizisidir.

Programlama Nedir ?

Programlama, “bilgisayara çeşitli görevleri yerine getirmesi için talimat vermenin” bir yoludur. Bu talimatlar, makinelerin nasıl çalışacağını veya çalıştıracağını söyleyebilir. Programlama bilgisayar ne yapması gerektiğini söyleyen, problemlere çözümler üreten ve bu çözümleri algoritmalar ile destekleyen bir öğretme işlemidir.

Günümüzde bir çok özelliğe sahip, işlem güçleri yüksek bilgisayarlar veya donanımlar mevcut. Hatta süper bilgisayar olarak adlandırılan milyon dolarlar harcanan donanımlar bile kullanılıyor. Ama bu bilgisayarlara ne yapacağını söylemediğiniz sürece, dünyanın en vasıfsız cihazlarından bir farkları olmayacaktır. Bilgisayarları hayatımızda bu kadar önemli hale getirmek için programlama tanımı devreye girmektedir.

Algoritma Nedir ?

Algoritma belirli bir problemi çözmek veya belirli bir şartı sağlamak için tasarlanmış yoldur .Bilgisayar bilimlerinde ve Matematikte bir işi yapmak için oluşturulan, başlangıç ve bitiş noktası olan sonlu işlemler kümesidir. Bilgisayar bilimlerinin önemli bir parçası olup , programlamada kullanılır. Tüm programlama dillerinin temelleri algoritmaya dayanır.

Bir bilgisayarları harekete geçirmek için, bilgisayar programları yazmak gerekmektedir. Bir program yazmak için, bilgisayar adım adım, tam olarak ne yapmak istediğini söylemek gerekir. Bilgisayar daha sonra bu adımları izleyerek istediğimiz eylemi gerçekleştirir.

Bilgisayara ne yapacağınızı söylediğinizde, bunun nasıl yapılacağını da seçersiniz. İşte burada bilgisayar algoritmaları devreye giriyor. Algoritma işi yapmak için kullanılan temel tekniktir. Algoritma kavramını anlamanıza yardımcı olacak bir örnek izleyelim.

Başka şehirden bir arkadaşınız sizi ziyarete gelecektir. Arkadaşınızın havaalanından evinize gelmesi gerektiğini varsayalım. İşte arkadaşınıza evinize gelmesi için verebileceğiniz dört farklı algoritma :

Taksiye bin

Araç kirala

Otobüse bin

Ben seni alırım

Bu algoritmaların dördü de tamamen aynı hedefe ulaşır, ancak her algoritma bunu tamamen farklı bir şekilde yapar. Her algoritmanın farklı bir maliyeti ve farklı bir seyahat süresi vardır. Örneğin bir taksiye binmek muhtemelen en hızlı, ama aynı zamanda en pahalı olanıdır. Otobüse binmek kesinlikle daha ucuz, ama çok daha yavaş. Algoritmayı koşullara göre seçersiniz.

Bilgisayar programlamada, verilen herhangi bir görevi yerine getirmenin birçok farklı yolu vardır. Her algoritmanın farklı durumlarda avantajları ve dezavantajları vardır. Bizler en uygun algoritmaları seçmeye çalışırız.

Programlama Dili Nedir ?

Dil, binlerce yıldır insanlar için iletişim aracı olmuştur. Bir topluluk için, dil insanların iletişim kurması gereken kelimeleri içeriyordur. Bilgisayarlara baktığımızda da , bu durumun çok farklı olmadığını göreceksiniz. Birbirleriyle iletişim kurması gereken bir çok donanım ve yazılım bileşenleri vardır.

Eski bilgisayarlar aslında devre ve kabloların yerleriyle oynayarak bir ve sıfırların elle değiştirilmesiyle programlandı. Tabii ki, bu tür ilkel programlar için kullanılan yöntem zordu. Bu nedenle programlama dillerinin oluşturulması, bilgisayar bilimlerini başka bir seviyeye taşıyan devrim niteliğinde bir adımdı. Normal dillerden farklı olarak, programlama dillerindeki anahtar kelimeler sınırlıdır. Bu kelimeleri birleştirerek , programcılar farklı türlerde programlar oluşturabilirler. Yazılan kodları makinenin anladığı dile dönüştüren ara birimlerde mevcuttur.

"Programlama Dili , insanların bilgisayarlarla etkileşime girdiği bir dizi talimattır."

Sözdizimi (Syntax) Nedir ?

Syntax (Sözdizimi ), yazılan herhangi bir sözün sırasıyla ilgilenen bilim dalıdır. Türkçede cümlelerin ögelerinin diziliminin bir sırası söz konusuysa diğer programlama dillerinde de aynı şey söz konusudur. Programlama dillerinde de tıpkı bizim günlük hayatta kullandığımız dilde olduğu gibi belli dizilim kuralları vardır. Bu dizilim kuralları sayesinde her şey makineler ve insanlar tarafından daha rahat anlaşılabilir ve yorumlanabilir bir hal almaktadır. Bunların dışına çıkıldığında veya bir eksiklik yapıldığında o kod satırı tam olarak yorumlanamaz

Java Nedir ?

Java Tarihçesi

1991 yılında Sun Microsystems şirketi mühendislerinden James Gosling ve 12 arkadaşı Green Project(Yeşil Proje) isimli bir proje geliştirmeye başladılar. Gömülü Sistemler üzerine çalışan James Gosling ve ekibi başlangıçta C ve C++ dillerini kullansalar da bu dillerin geliştirdikleri projeye uygun bir dil olmadığını ve geliştirdikleri proje için yetersiz olduğunu görüp yeni bir arayış içine girdiler ve bu sırada “Oak” şimdiki adıyla Java dili doğmuş oldu.

Java, doğrudan C++ ile bağlantılıdır. C++ ise C’nin devamıdır. Java, karakteristik özelliklerinin birçoğunu bu iki dilden almıştır. C’nin sözdizimi (syntax), C++’ın ise nesne yönelimli programlama (object oriented programming- OOP) kavramları Java’ya miras kalmıştır. Bunun yanı sıra, Java, bu dillerdeki karmaşıklığı ortadan kaldırmış, eksik özelliklerini ise tamamlamıştır.

Java'nın Avantajları Nelerdir ?

Basit Olması

Java’nın yazımı kolaydır. daha okunaklı bir yapısı vardır ve dikkat çekicidir. Java geleceğe dönük olarak oluşturulacak yapılarla kullanımı kolaylaştırılır, sadeleştirilir ve öğrenmesi kolaydır.

Tarafsız Mimariye Sahip

Java özel makinelere ya da işletim sistemi mimarilerine bağlı değildir. Java donanımdan bağımsızdır. Java'yı platform bağımsız kılan özelliği yazılan kaynak kodlar derlendikten sonra ara bir dil olan byte code'a çevrilmesidir. Ara dile çevrilen bu kod parçaları Java Virtual Machine vasıtasıyla yorumlanır ve çalıştırılır. Buradaki tek sorun hız sorunudur. Çünkü işletim sistemiyle birlikte Virtual Machine’de bilgisayarın kaynaklarını kullandığı için daha yavaş çalışmaktadır.

Nesne Yönelimli Olması

Java da C++ gibi nesne yönelimlinin özelliklerinden faydalanır. Sınıflar oluşturularak daha az kod yazıp daha fazla iş yaparak kod tekrarı önlenmiş olur. Böylece geliştirme sürecinin verimliliği artar.

Zengin Kütüphane Desteği

Java çok zengin ve geniş bir kütüphaneye sahiptir ve ücretsiz olarak erişim sağlayıp temin edilebilir. Bu sayede bir çok platformda uygulama geliştirme imkanı sağlar.

Multi-Thread & Dinamik

'Multi-Thread' bir programda bir çok işlemin aynı zamanda gerçekleşmesi özelliğidir. Eş zamanlı işlemleri programlayabilmenize imkan tanır. Java bu tür program geliştirmeye destekler.

Neden Java Kullanmalıyız ?

Bilgisayarlarının %97'sinde Java Bulunuyor.

Dünya’da 9 Milyon Java Geliştiricisi Var

3 Milyar Cep Telefonunda Java Bulunuyor

125 milyon TV cihazı Java kullanıyor

Dizüstü bilgisayarlardan veri depolama merkezlerine, oyun konsollarından bilimsel süper bilgisayarlara, cep telefonlarından Internet'e kadar Java her yerde!

Java ile Uygulama Geliştirme Yapabileceğiniz Alanlar

Teknoloji ve onları geliştirmek için kullanılan diller sürekli bir evrim geçirmekte. Java dili, çok sayıda uygulamaya sahip popüler programlama dillerinden biridir. Java ile ne tür uygulanalar yapılabilir ?

Mobil Uygulamalar : Java, mobil uygulama geliştirme için çok kullanılan bir programlama dilidir. Android Studio ve Kotlin gibi yazılımlarla uyumludur. Çünkü Android İşletim Sistemi Java ile yazılmıştır. Android'in sınıf dosyalarını yürütmek için DVK (Dalvik Virtual Machine) kullanırken Java Virtual Machine (JVM) üzerinde çalışmaktadır. Java ve OOP ilkeleri ile Android ile daha iyi güvenlik ve basitlik sağlar.

Masaüstü Uygulamalar : Masaüstü uygulamaları Java ile kolaylıkla geliştirilebilir. Java ayrıca AWT, Swing ve JavaFX gibi eklentileri ile arayüz geliştirme desteği ve kolaylığı sağlar.

Web Tabanlı Uygulamalar : Java, web uygulamaları geliştirmek için de kullanılır. Servlets, Struts veya JSP eklentileri ile web uygulamaları için geniş destek sağlar. Bu teknolojiler sayesinde ihtiyacınız olan her türlü web uygulamasını geliştirebilirsiniz.

Kurumsal Uygulamalar : Java en çok talep gören programlama dillerden biridir. Büyük ve küçük işletmeler, bu dilde yazılmış uygulamalarla desteklenmektedir. Bu, yenilikçi bir ticari yazılım türü fikriniz varsa, Java muhtemelen sizin için en iyi dildir demektir. Havayollarının, devlet kurumlarının her gün kullandığı özel işletme yazılımlarının çoğu için de geçerlidir.

Bilimsel Uygulamalar : Bilimsel hesaplamalar ve matematiksel işlemler yaptırmak için Java çok kullanılan bir dildir. MATLAB gibi en popüler uygulamalardan bazıları, GUI (Arayüz) ve çekirdek yapılarında Java kullanır.

Oyun Programlama : Java, bağımsız oyun geliştirme şirketleri tarafından ve mobil oyunlar oluşturmak için yaygın olarak kullanılmaktadır. Java, 3 boyutlu oyunların tasarımı söz konusu olduğunda benzersiz bir yeteneğe sahip olan açık kaynaklı 3D Motor olan jMonkeyEngine'in desteğine sahiptir. Dünyanın en iyi mobil oyunlarının çoğu Java'da geliştirilmiştir. Minecraft, Mission Impossible III ve Asphalt 6 vb.

Gömülü sistem uygulamaları geliştirebilirsiniz. (Java2ME Embedded altyapısı ile) Java, gömülü sistem tasarımcıları için bir dizi avantaja sahiptir. Java, dil olarak, C ++ tehlikeleri olmadan nesne yönelimli programlamaya izin verir. Örneğin, Java sınıf kalıtımına izin verir, ancak birden çok ebeveynden gelmez, bu nedenle kafa karışıklığı fırsatı yoktur. Ayrıca Java, teknoloji geliştiricilerinin değişen pazar koşullarına hızla yanıt vermesini mümkün kılar. Uygun şekilde uygulanırsa, gömülü bir uygulamanın Java bölümleri uygulamanın geri kalanından izole edilebilir, böylece görev açısından kritik bölümleri Java bölümlerinin yol açtığı bozulmaya karşı korur.

Java’da Yazım ve İsimlendirme Kuralları

Yazım Kuralları

* **Türkçe Karakter Kullanımı** : Java ve bir çok programlama dili Türkçe karakter desteklememektedir. Programda ki metin ifadeleri dışında Türkçe karakter kullanılmamalıdır. Türkçe Karakterler : ç, ı, ü, ğ, ö, ş, İ, Ğ, Ü, Ö, Ş, Ç
* **Harf Duyarlılığı** : Java harfe duyarlı bir dildir. Kelimelerdeki küçük ve büyük harfler farklı olarak algılanır. Java’da Kodluyoruz ile kodluyoruz farklı anlama gelmektedir.
* **Sınıf Adları** : Java’da sınıf adlarının ilk harfi büyük olmalıdır. Eğer 2 kelimeyi birleştirerek bir sınıf adı oluşturursak bu isimlerin baş harfleri büyük olmalıdır. Örnek: CamelCase
* **Metot Adları** : Metot adları küçük harfle başlar. Metot adı verilirken iki kelime birleştirilecekse ismin başlangıç harfi küçük diğer birleştirilen kelimelerin başlangıç harfleri büyük yazılır. Örnek: camelCaseOrnek

İsimlendirme Kuralları

Camel Case Nedir ?

Camel Case, bir bileşik sözcük içindeki her kelimenin ilk sözcük dışında ki sözcüklerin ilk harfleri büyük harflerle yazıldığı bir adlandırma kuralıdır. Yazılım geliştiricileri genellikle kaynak kodu yazarken "**Camel Case**" kullanır. **Camel Case** kullanımı zorunlu bir kullanım olmasa da yazılım dilinin jargonudur ve genelde tüm yazılımcılar bu kurala uyar. Bu kural sayesinde değişken isimleri daha okunur olur.

**Örnek** : patikaDev, camelCaseKurali, javaPatikasiBasliyor

Upper Camel Case Nedir ?

Upper Camel Case isimlendirme kuralı ise, bileşik bir sözcükteki tüm sözcüklerin ilk harflerinin büyük olmasıdır.

**Örnek** : PatikaDev, UpperCamelCaseKurali, JavaPatikasiBasliyor

Snake Case

Kelimeler **alt tire** (\_) ile birbirine bağlanır.

Upper snake case **örnek**: Hello\_World

Lower snake case **örnek**: hello\_world

Screaming Snake Case

Bütün harfleri büyük yazılır. Genellikle sabit isimlendirmede kullanılır.

Java'da İsimlendirme Kuralları

* Java'da isimlendirilen tüm ögeler sadece A-Z veya a-z gibi harfler, $ karakteri veya \_ karakteri ile başlayabilirler.
* Keyword’ler (Yasaklı Kelimeler) isimlendirmede kullanılamaz



* Sınıflar için **upper camel case** kullanılır. **HelloWorld**
* Metotlar için **lower camel case** kullanılır. **helloWorld**
* Değişkenler için **lower camel case** kullanılır. **helloWorld**
* Sabitler için **screaming snake case** kullanılır. **HELLO\_WORLD**

# Main Metodu ve Kullanımı

Java'da bir program çalışacağı zaman, kodlar ilk olarak main metodu içerisinden başlar. Main metodu sayesinde derleyiciye, programı buradan başlatılması konusunda referans oluşturulur.

public class JavaPatika { public static void main(String[] args) { System.out.println("Hello World!"); } }

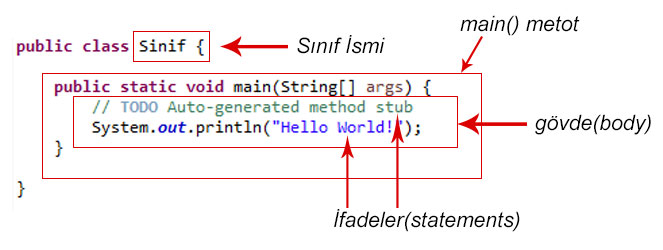
Artık programı çalıştırdığımızda, derleyici ilk olarak **main metodu** okuyup sonrasında gerekli işlemleri yapacaktır.

Main metodu yazılırken, args yerine başka bir isimlendirme yapılabilir. Ancak genellikle args sözcüğü kullanılır ve bu sözcük arguments sözcüğünün kısaltmasıdır. Arguments ile ifade edilen kısım, sınıf çalıştırılırken JVM tarafından bu sınıfa verilen parametrelerdir.

# Programlamaya Başlangıç

İlk öğreneceğimiz kod parçacığı, Java'da ekrana veri bastırma kodu olan **System.out.println("Java101");** komutunu öğreneceğiz. Ama ilk önce bilmemiz geren bir konu Java'da kodların nereye yazıldığı ve yazım kuralları ya da diğer adıyla söz dizimi (Syntax). Genellikle Syntax olarak İngilizce adıyla duyduğumuz bu terim herhangi bir programlama dilinin yazım kuralını belirler.

## Genel Sözdizimi (Syntax)



Yukarıda ki örnekte göreceğiniz gibi, en üst kısımda **"public class Sinif"** komutu ile bir sınıf (class) oluşturuyoruz. Java'da sınıflar içerisine kodlarımızı yazarız ve ilerleyen derslerde sınıflar konusuna derin bir giriş yapacağız. Daha sonra program çalıştığında çalışan bir metot olan "**Main Metodu**" sınıfımızın içerisine yazarız çünkü programı çalıştırmak için derleyici ilk olarak main metot içerisindeki komutları okuyacaktır.

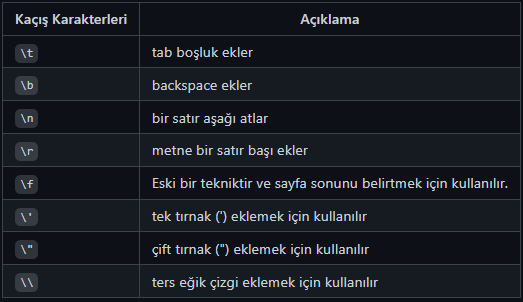
Main metot içerisine "**Gövde (body)"** adını veririz ve komutlarımızı ya da diğer adıyla ifadelerimizi gövde içerisine yazarız. Buraya yazdığımız kodlar derleyici tarafından yorumlanır ve çıktı olarak kullanıcıya verir. Genel olarak **Java'nın Genel Sözdizim** kuralları bu şekildedir.

## Ekrana Veri Yazdırma

Java'da ekrana veri yazdırmak için System.out.print("Hello World!") kod parçacığı kullanılır. Bu komuta baktığımızda, iki parantez arasında, çift tırnaklar arasına ekrana yazdırmak istediğimiz sözcüğü yazmalıyız. Bu komutun iki farklı kullanım şekli mevcuttur, yazılan komuttan sonra yeni satıra inilmesi isteniyorsa System.out.println(); kullanılırken aynı satırda kalınması isteniyorsa System.out.print(); şeklinde kullanılır.

## Escape Karakterler

Java'da Escape (Kaçış) karakterleri ile bazı özel durumlar durumlar gerçekleştirilir. Kaçış karakterleri ( / ) ters eğik çizgi ile ifade edilip sonrasında yazılan karakter ile özel işleve sahip olurlar.



# Yorum Satırları

Yorum satırları, kod içi belgeleme amacıyla kullanılan ve derleyiciler tarafından dikkate alınmayan kod parçalarıdır. Yorum satırları oluşturmamızın sebebi, Yazdığı kodun kritik kesimlerini açıklayarak, o koda daha sonra bakan kimselerin (büyük olasılıkla kendisinin) işini kolaylaştırmak amaçlanmaktadır. Java’da yorum satırları 3 farklı şekilde yazılır:

## // ile yapılan yorumlar

Tek satırlık bir açıklama yapılacaksa o satırın başına // işareti yazılır. ; // işaretinden sonra satır sonuna kadar her şey yorum olarak kabul edilir. Anlaşılacağı üzere bu işaretin satırın en başında olması zorunlu değildir. Ancak kodlama alışkanlığı bakımından satır başında kullanılması daha uygundur.

// bu bir yorum satırıdır int number = 10; // number değişkenine 10 değeri atandı.

## /\* ... \*/ ile yapılan yorumlar

Eğer birden fazla satırda yazılan bir açıklama varsa, her satırın başına // işareti koymak programcıya zor gelebilir. Bunun yerine, açıklama olarak değerlendirilmesi istenen satırlar /\* ve \*/ işaretleri arasına alınır. Bu iki işaret arasında kalan kesimler derleyici tarafından yorum satırı olarak kabul edilir.

/\* Birden fazla satırdan oluşan bir yorum satırlarıdır. Ancak yorumların bu yolla ifade edilmesi için birden fazla satırdan oluşması zorunluluğu yoktur. \*/ int number = 10; // number değişkenine 10 değeri atandı.

## /\*\* ... \*/ ile yapılan açıklamalar

Bir uygulama geliştirilirken kod içi belgeleme yapmak güzel bir programlama alışkanlığıdır. Çünkü hem yapmakta olduğunuz işi en güzel o işi yaparken açıklayabilirsiniz, hem de açıklayabildiğiniz kodu anlamışsınız demektir ve o kodu açıklayarak yazdığınız için hata yapma olasılığınız düşer.

Öte yandan, çoğu zaman uygulamaların raporlarının oluşturulması gerekir. Kod yazıldıktan sonra kodun içine yazılan açıklamalardan bir belge oluşturarak bu belgeyi raporun sonuna eklemek programcının yükünü hafifletecektir. İşte şimdi bahsedeceğimiz üçüncü yöntem bu amaçla kullanılır. /\*\* ve \*/ işaretleri arasına yazılan açıklamalar bir takım özel etiketler içerebilir. Kod içi belgeleme, bu etiketleri tanıyan ve etiketlerden faydalanarak belge üreten bir aracın yardımı ile belgeye dönüştürülebilmektedir.

Bu tarzda yazılan açıklama satırlarına Javadoc adı verilmektedir. Javadoc için kullanılabilecek bazı örnekler ve ne için kullanılabilecekleri aşağıda listelenmiştir:



Değişkenler ve Veri Tipleri

Değişkenler programlamada geçici bilgileri sakladığımız ve programcı tarafından belirlenen yapılardır. Değişkenler programlamanın temel yapısını oluşturmaktadır ve tüm programlama dillerinde kullanılan bir yapıdır. Değişkenler sayesinde program içinde yaptığımız işlemleri hafıza da tutar ve gerektiği yerlerde kullanırız. Değişkenlerin 4 özelliği bulunur ; **Veri Tipi, İsim, Değer ve Adres.**

Değişkenlerde **Veri Tipi**, adından da anlaşılacağı üzere verinin saklanacağı türünü belirtmektedir bunlar **sayılar, sözcükler ve programatik** alanlar olabilir. Her değişkenin program içinde kullanıldığı ve çağrıldığı benzersiz bir **ismi** vardır. Bu değişkenlere birde **değer** atarız , işletim sistemimizde bu değişkeni hafızada tutar ve bir **adres** belirler.

Java'da değişkenlerin veri tipleri vardır. Bu tipler Java'da varsayılan olarak tanımlı gelen ilkel tipler (primitive) de olabilir yahut yazılımcıların kendi tanımladığı ilkel olmayan tipler (non-primitive) de olabilir. İlkel tipler her zaman bir değere sahiptir. İlkel olmayan tipler ise 'null' (boş) olabilir. İlkel olmayan türlerin tümü aynı boyuta sahipken ilkel veri tiplerin boyutu alacağı veri tipine bağlıdır.

Java Değişken Tanımlama

<veri tipi> <değişken ismi> = veri (değer)

İlk önce değişkenin veri tipini ve değişkenin ismini yazarız ve istenirse aynı matematikteki gibi "=" eşittir ile değerini atarız.

int number; // number isminde, int veri tipinde bir değişken tanımlanmış

Veri tipleri aynı olan değişkenleri aynı satırda tanımlayabiliriz

int a, b, c; // int veri tipinde 3 tane değişken tanımlanmış

Değişkeni tanımladıktan sonra, atama operatörü (=) kullanarak değişkene atayabiliriz.

double pi; // ilk başta double türünde bir değişken tanımladık pi = 3.14; // Daha sonra bu değişkene bir değer atadık

Eğer bir değişkene hemen değer atayacaksanız, bunu iki satırda yapmak yerine tek bir satırda halledebilirsiniz.

double pi = 3.14;

Aynı satırda aynı türden birden fazla değişken tanımlıyorsak :

int a = 1 , b = 2; // Aynı satırda int türünde 2 farklı değişken tanımlanmış ve ikisine de değer verilmiş

Değişkene verilen değer sonrasında değiştirilebilir, ama aynı isimde ikinci bir değişken oluşturulamaz ve hata alırız.

**Hatalı Kullanım :**

int a = 5; // a isminde bir değişken tanımlanmış int a = 1 ; // Bu satır hataya neden olur, a değişkeni zaten var

**Doğru Kullanım :**

boolean a = true; // a isminde bir değişken tanımlanmış ve varsayılan bir değer verilmiş a = false; // a değişkeninin değeri değiştirilmiş

Java'daki İlkel Veri Tipleri

* Tam Sayılar
* Byte
* Short
* Integer
* Long
* Ondalıklı Sayılar
* Float
* Double
* Karakterler
* Char
* Mantıksal Değerler
* Boolean

# Byte, Short, Int ve Long Veri Tipleri

Java'da tam sayıları belirten veri tipleri **Byte, Short, Integer ve Long**'tur.

Byte

* 8 bit uzunluğundadır. Max 127 , Min -128 değerleri arasındadır.
* Anahtar sözcük : **byte**

Short

* 16 bit uzunluğundadır. Max 32,767 , Min -32,768 değerleri arasındadır.
* Anahtar sözcük : **short**

Integer

* 32 bit uzunluğundadır. Max 2,147,483,647 , Min -2,147,483,648 değerleri arasındadır.
* En çok tercih edilen veri tipidir , sebebi ise optimize uzunluktadır.
* Anahtar sözcük : **int**

Long

* 64 bit uzunluğundadır. Max 9,223,372,036,854,775,807 , Min -9,223,372,036,854,775,808 değerleri arasındadır.
* Int’in yetersiz olduğu yerlerde kullanılır
* Anahtar sözcük : **long**

public class JavaPatika { public static void main(String[] args) { byte a = 120;

short b = 1000;

int c = 100000;

long d = 10000000;

} }

Float ve Double Veri Tipleri

Java'da ondalıklı yani küsuratlı sayıları belirten veri tipleri **Float** ve **Double**'dır.

Float

* 32 Bit boyutundadır ve **1.4×10^-45** ile **3.4×10^38** aralığında bir değer tanımlanabilir.
* Float içerisine tam sayı yazdığımız zamanda bile o sayı 1.0 şeklinde ondalıklı olarak algılar.
* Float ile double ayırmak için , float tanımlamalardan sonra ‘f’ veya ‘F’ konulmalıdır.
* Anahtar Sözcük : **float**

Double

* 64 Bit boyutundadır ve 4.9×10^-324 ile 1.8×10^308 aralığında bir değer tanımlanabilir.
* Üst düzey matematiksel işlemlerde kullanılır
* Anahtar Sözcük : **double**

Ondalık Sayı Veri Tiplerinden Hangisi Tercih Edilmelidir ?

Bu sorunun cevabı değişken olmakla beraber hangi durumlarda Double veya Float kullanımınız için dikkat etmeniz gereken hususlar

* **Double** tipi, yüksek duyarlıklı ve hassas matematiksel işlemlerde kullanılır.
* **Float** basittir : hız ve bellek.
* **Double** daha yavaş ve fazla yer kaplar.(Bu işlemler arasında milisaniye oynar)
* **Java** hassas matematiksel işlemler, örneğin sinüs kosinüs fonksiyonları, **double** tipi değer döndürür.

public class JavaPatika { public static void main(String[] args) { float number1 = 3.14F; float number2 = 3.14f; double number3 = 3.14; } }

Char ve Boolean Veri Tipleri

**Char**

Java'da karakter değişkenleri saklamak için **Char** kullanılır. Char veri tipleri birleşerek **String** Sınıfından bir yapıya dönüşür.

* Karakterler **Char** ile saklanır.
* Diğer dillere göre **Char** Java’da 16 bittir.
* Java **Unicode** karakter setini kullanır ve tüm dilleri içerir.
* Java evrensel bir dil olarak tasarlandığı için karakter seti de evrensel set olan **Unicode** ile tanımlanmıştır.
* Anahtar Sözcük : **char**

Boolean

Java, mantıksal değerleri saklamak için **boolean** adında bir tipe sahiptir.

* Boolean sadece **iki** değer alabilir : **True ve False**
* Genellikle **koşul ve döngü** işlemlerinde, kontrol amaçlı olarak kullanılır.
* Anahtar Sözcük : **boolean**

public class JavaPatika { public static void main(String[] args) { char letter = 'u'; boolean logic1 = true; boolean logic2 = false; } }

# String Sınıfı

**String** sınıfı **java.lang** kütüphanelerinde bulunan ve metinlerle ilgili her türlü işlemin yapıldığı sınıftır. Java'da genellikle kelime tutmak için char yerine String sınıfı kullanılır. Basitçe şöyle düşünebiliriz , Char veri tipi tek bir karakter tutabiliyorken, charların birleşmesiyle oluşan sözcükleri String Sınıfı tutmaktadır.

public class JavaPatika { public static void main(String[] args) {

String words = "Hello World";

} }

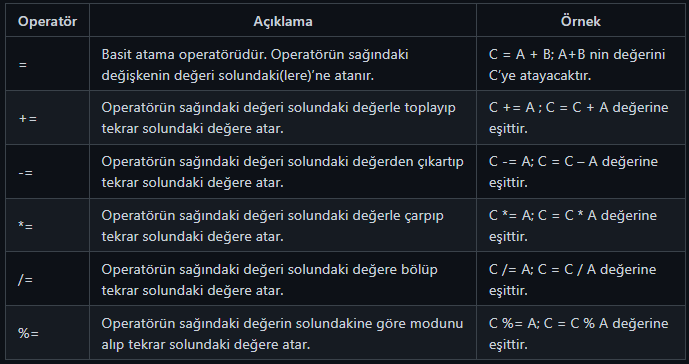
Temel Operatörler

Java dilinde operatörler birçok işlemi yapabilmenize olanak tanır. Örneğin: matematiksel operatörlerle birlikte aritmetik işlemler yapabilmenizi, ilişkisel operatörlerle verileri kıyaslayabilmeyi, atama operatörleri ile değişkenlerin değerlerini değiştirmeye fırsat verir.

Java'da operatörler aşağıdaki gibi listelenebilir:

* Atama Operatörleri
* Aritmetiksel Operatörler
* İlişkisel ve Eşitlik Operatörler
* Koşul Operatörler
* Mantıksal Operatörler

Atama Operatörü



Aritmetik Operatörler

Java'da Aritmetik Operatörler adından da anlaşılacağı üzere matematiksel işlemleri programlama dilinde uygulamamızı sağlarlar.

* Toplama : a + b
* Çıkarma : a – b
* Çarpma : a \* b
* Bölme : a / b
* Mod alma : a % b
* 1 arttırma : a++
* 1 eksiltme : b--

Karşılaştırma Operatörleri

Java'da Karşılaştırma Operatörleri iki nesnenin birbirleriyle olan durumlarını belirler.

* Eşitlik : a == b
* Eşit Değil : a != b
* Büyüktür : a > b
* Küçüktür : a < b
* Büyük Eşittir : a >= b
* Küçük Eşittir : a <= b

Mantıksal Operatörler

Java'da Mantıksal Operatörler , nesnelerin veya ifadelerin mantıksal değerlerini yansıtır.

* Ve : a && b
* Veya : a || b
* Değil : !(a&&b)

Koşul Operatörü

Java'da Koşul Operatörleri ifadelerin sonucunda oluşacak olayları belirler.

* a = 5 ;
* b = (a == 1) ? 1 : 0
* Çıktısı : 0

# Kullanıcıdan Veri Alma

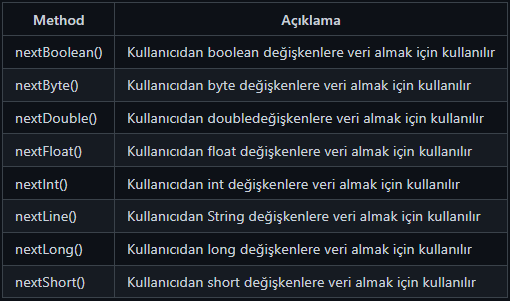
Java’da **kullanıcıdan veri almak** için **Scanner** sınıfı kullanılır. Ama bu sınıfı kullanmadan önce kodumuza **Scanner** sınıfını dahil etmemiz gerekir. Bunun için **import** deyimi kullanılır ;

import java.util.Scanner;

İmport deyimi projenin en başına yazılır. Kullanıcıdan verileri almak için **değişkenlere** ihtiyacımız vardır. İlk önce **"a"** adında veri tipi **integer** olan bir değişken oluşturalım. Oluşturduğumuz **"a"** değişkenine veriyi kullanıcıdan almak için yapmamız gereken **Scanner** sınıfını kullanmak. **Scanner** sınıfından türeyen adı **"input**" olan bir nesne tanımlayalım. Sınıf ve Nesne kavramları ilerleyen derslerde detaylıca anlatılacaktır. **Scanner** sınıfından nesne ürettikten sonra değişkenimize veri almak için, değişkenimizin türüne göre bir kod yazmamız gerekecektir. Eğer değişkenimizin "**integer**" türünde ise **"input.nextInt()"** veya **double** türünde ise "**input.nextDouble()**" kod bloğu kullanılmalıdır.

import java.util.Scanner; public class JavaPatika { public static void main(String[] args) { Scanner input = new Scanner(System.in); int a,b; System.out.println("A sayısını giriniz : "); a = input.nextInt(); System.out.println("B sayısını giriniz : "); b = input.nextInt(); System.out.println("A Sayısı : " + a); System.out.println("B Sayısı : " + b); } }

## Veri Tiplerine Göre Scanner Metotları



import java.util.Scanner; public class JavaPatika {public static void main(String[] args) { Scanner inp = new Scanner(System.in); // String ÖrneğiString adSoyad = inp.nextLine(); // Sayı Örnekleriint yas = inp.nextInt(); double maas = inp.nextDouble(); // Çıktılar System.out.println("Ad Soyad: " + adSoyad); System.out.println("Yaş : " + yas); System.out.println("Maaş : " + maas); } }

# Karşılaştırma Operatörleri

Java'da **Karşılaştırma Operatörleri** (Java Comparison Operators) sayesinde, farklı iki değişkenin birbirleriyle olan durumları hakkında bir yorum yapabilirsiniz.

Kısacası Java'da **Karşılaştırma Operatörleri** , bir değişkenin başka bir değişkenden **büyük, küçük, eşit değil veya eşit olup** olmadığını belirler. Bu operatörlerin çoğunu günlük hayatımızda ve matematikte kullanıyoruz. İki değişkenin eşit olup olmadığını test ederken **"=" değil de "=="** kullanmanız gerektiğini **(yani çift eşittir)** unutmayın.

## JAVA KARŞILAŞTIRMA OPERATÖRLERİ

**== Eşittir**( x == y )

**!= Eşit Değildir** ( x != y )

**> Büyüktür** ( x > y )

**< Küçüktür** ( x < y )

**>= Büyük Eşittir**( x >= y )

**<= Küçük Eşittir** ( x <= y )

## Örnek

Operatörlerden bahsetmeden önce 4 adet değişken tanımlayıp,bunlara değer ataması yapalım ve ilişkisel ifadelerimizi bu değişkenler üzerinden karşılaştıralım. \*\*Bu karşılaştırmaları yaparken matematiksel ifademizi cümle haline getirip bu cümlenin doğruluğunu ve yanlışlığını test edeceğiz.\*\* int A = 10; int B = 20; int C = 10; int D = 40 ;

Elimizde A,B,C ve D olmak üzere 4 adet değişkenimiz var. Şimdi değişkenlerin birbirleri ile olan ilişkilerini inceleyelim.

### == (Eşitlik Operatörü)

(A == B) ;//Matematiksel ifademizi cümle haline getirelim. (10 Eşittir 20) bu cümle yanlış bir cümledir.10 sayısı 20 sayısına eşit olamayacağından sonucumuz yanlış yani false'tur. Aynı cümleleri diğer matematiksel ifadelerimiz içinde yazalım. (A == C) ; // (10 Eşittir 10 ) evet bu cümle doğru sonucumuz true. (C == D) ; // (10 Eşittir 40 ) bu cümle yanlıştır ve cevabımız false.

### != (Eşit Değil Operatörü)

(A != D) ; // (10 Eşit Değildir 40 ) Evet 10 sayısı 40'a eşit değildir ve sonucumuz true'dur. (A != C) ; // (10 Eşit Değildir 10 ) Bu cümle doğru değildir.Bu yüzden cevabımız false'tur. (C != B) ; // (10 Eşit Değildir 20 ) True.

### > (Büyüktür Operatörü)

(A > D) ; // (10 Büyüktür 40 ) 10 sayısı 40 sayısından büyük değildir cümle yanlış olduğundan cevabımız false olacaktır. (D > C) ; // (40 Büyüktür 10) 40 sayısı 20 sayısından büyüktür,ifade doğrudur sonucumuz true'dur. (C > B) ; // (10 Büyüktür 20) False

### >= (Büyük Eşittir Operatörü)

(A >= D) ; // (10 Büyük Eşittir 40 ) 10 sayısı 40 sayısından büyük veya eşit değildir cümle yanlış olduğundan cevabımız false olacaktır. (A >= C) ; // (10 Büyük Eşittir 10) Burada 10 sayısı 10 sayısından büyük değildir ancak ona eşit olduğundan sonucumuz true olacaktır. (C >= B) ; // (10 Büyük Eşittir 20) False

### < (Küçüktür Operatörü)

(A < D) ; // (10 Küçüktür 40) 10 sayısı 40 sayısından küçük ve cümle doğrudur.Sonucumuz true. (D < C) ; // (40 Küçüktür 10) Bu cümle yanlış olduğundan sonucumuz false olacaktır. (C < B) ; // (10 Küçüktür 20) True

### <= (Küçük Eşittir Operatörü)

(A <= D) ; // (10 Küçük eşittir 40) 10 sayısı 40 sayısından küçüktür ve cümle doğrudur.Sonucumuz true. (A <= C) ; // (10 Küçük eşittir 10) 10 sayısı 10 sayısından küçük değildir ancak eşit olduğundan sonucumuz true. (C <= B) ; // (10 Küçük eşittir 20) True

// Aşağıdaki Kodları kendi IDE'nizde çalıştırarak,farklı değerlerle sonuçları test edebilirsiniz. int A = 10; int B = 20; int C = 10; int D = 40 ; // == Operatörü System.out.println(A == B); System.out.println(A == C); System.out.println(C == D); // != Eşit Değil Operatörü System.out.println(A != D); System.out.println(A != C); System.out.println(C != B); // > Büyüktür Operatörü System.out.println(A > D); System.out.println(D > C); System.out.println(C > B); // >= Büyük-Eşittir Operatörü System.out.println(A >= D); System.out.println(A >= C); System.out.println(C >= B); // < Küçüktür Operatörü System.out.println(A < D); System.out.println(D < C); System.out.println(C < B); // <= Küçük-Eşittir Operatörü System.out.println(A <= D); System.out.println(A <= C); System.out.println(C <= B);

# Kod Blokları (Scope)

İki küme parantezi “{“ ve “}” arasında kalan kod kesimine blok (scope) denir. “{“ işareti bir kod bloğu başlatır ve “}” işareti başlatılan kod bloğunu bitirir. Bu işaretler kodun sınırlarını belirlemekte kullanılır. Herhangi bir değişken, tanımlandığı kod bloğu içinde fiziksel olarak vardır ve o kod bloğu içine yazılan kod kesimlerinden erişilebilirdir. Bir blok içinde aynı değişken adı birden fazla kez kullanılamaz.

public class JavaPatika {// JavaPatika sınıfına ait alan public static void main(String[] args) { // Main metota ait alan System.out.println("Burası main metota ait"); } }

Yukarıda ki örnekte göreceğiniz üzere, kod blokları kodlar arasındaki hiyerarşik düzeni de belirtir.

Bu durumda ;

- System.out.println("Burası main metota ait"); => main metoda ait - public static void main(String[] args) {...} => JavaPatika sınıfına ait

İç içe bir kod yapısını ve ögelerin bir birilerine aitlik durumlarını kod blokları ile sağlarız. Ayrıca her kod bloğundan sonra bir tab boyutunda boşluk bırakılır. Bunun sebebi kodun okunabilirliğini arttırmaktır. Kodun okunabilirliği bir yazılımcı için çok önemli bir noktadır.

### Kötü Kod Örneği

public class JavaPatika { // JavaPatika sınıfına ait alan public static void main(String[] args) { // Main metota ait alan int a=1,b=2,c=3; System.out.println("Burası main metota ait"); } }

### Temiz Kod Örneği

public class JavaPatika {// JavaPatika sınıfına ait alan public static void main(String[] args) { // Main metota ait alanint a = 1, b = 2, c = 3; System.out.println("Burası main metota ait"); } }

If ve Else Blokları

Java'da mantıksal ve koşullu ifadeler için **if** blokları kullanılır. Java 'da **if** blokları Türkçede "**eğer**" cümlesinin verdiği anlamla aynı şeyi ifade etmektedir. Programlamada koşul oluşturmak için **if ve else** bloklarını kullanırız. **Else** deyimi **"değil ise"** anlamı katmaktadır ve **if** deyiminden sonra kullanılır. Programlama da **if ve else deyimleri** çok önemlidir ve kodlamanın temel yapı taşlarını oluştururlar. If ve Else blokları ile programımız içerisinde ki algoritmalarımızı oluştururuz.

Koşula göre program içinde farklı işlemleri yerine getirmek gerekebilir. If-else muhtemelen yazılım programlarında en yaygın kullanılan karar mekanizmasıdır. Bu tarz durumları Java'da kodlayabilmek için if-else karar mekanizmasının yanında switch-case gibi yapılarda kullanılmaktadır. İf-else karar mekanizması istenilen koşul gerçekleştiği takdirde çalışır. Kısacası belirlenen koşul doğruysa yazılan kod çalışır.

Java'da Karar Mekanizmaları

Java'da koşullu ifade ve şart oluşturmak için i**f ve else** kullanılırız. Ayrıca **else if, switch** gibi koşullu ifadelere sahiptir.

* Programımızda belirtilen bir koşul doğruysa yürütülecek bir kod bloğu belirtmek için **if** kullanılır.
* Aynı koşul yanlışsa yürütülecek bir kod bloğu belirtmek için **else** deyimi kullanılır.
* İlk koşul yanlışsa, test edilecek yeni bir koşul belirtmek için eğer **if else** deyimi kullanılır.
* Yürütülecek birçok alternatif kod bloğu belirtmek için **switch** deyimi kullanılır.

Örneğin: "yaşı 50'den küçük olanların personel kayıtlarını getir" gibi bir ifadede yaşı 50'den küçük olanları tespit etmek için karar mekanizmaları kullanılır. Eğer koşul sağlanmıyorsa başka bir kod bloğu işletilir. İfadenin yazılımsal olarak Java'da karşılığı aşağıdaki gibidir.

if(age < 50) { // personel kayıtlarını getir }

Koşulun gerçekleşip gerçekleşmediği "if" cümlesi içindeki ifadeye bağlıdır. Eğer mantıksal ifade true ise "if" kod bloğu çalıştırılır. Tabi ifadenin false döndüğü durumda da programın bir şeyler yapmasını isteyebiliriz. Bu durumda ise "else" ifadesi kullanılır. Aşağıdaki gibi bir örnekle açıklayabiliriz.

Eğer, faiz oranı %70'den büyükse "Kurumsal Müşteri" tipinde kredi ver, değilse "Standart Müşteri" tipinde kredi ver şeklinde bir karar mekanizması aşağıdaki gibi tasarlanabilir.

if(creditRatio > 0.7) { System.out.println("Kurumsal müşteri tipinde kredi"); } else { System.out.println("Standart müşteri tipinde kredi"); }

Koşullar birden fazla olabilir ve hiçbir koşula uymuyorsa en sonunda varsayılan bir duruma girilir ve o kod bloğunu çalıştırmak gerekir. Bu tarz durumlarda ise "if - else if - else" gibi yapılar kullanılır. If-else yapısına bir örnek daha aşağıdaki gibi olabilir.

if(ortalama >= 50){ System.out.println("Dersi Geçtiniz"); }else{ System.out.println("Dersten Kaldınız"); }

Bu kod bloğunda karar yapımda eğer ortalama 50 ve üstü ise Dersi Geçtiniz yazar, değilse Dersten Kaldınız yazar. if kısmı koşulun doğruluğunu kontrol ediyor, else kısmı ise koşul yanlış ise devamında yazan kodu çalıştırıyor. Yani eğer ilk kısımda TRUE dönerse Dersi Geçtiniz yazacak, FALSE dönerse else kısmındaki kod çalışacak.

else if

**if - else - if**

if-else-if kalıbında, bir if satırı, istediğiniz sayıda else-if satırı ve isteğe bağlı olarak tanımlanan bir else satırı bulunur. Yani, else satırının tanımlanması şart değildir. Program, en başta yer alan if satırı ile if-else-if kalıbını değerlendirmeye başlar. İlk doğru ifadenin bulunduğu if veya else-if satırı ile karşılaştığında ilgili işlem satırını çalıştırır. Kalıbın geri kalan tüm satırlarını değerlendirmeye almadan geçer. Eğer if ve else if satırlarında yer alan ifadelerden hiç biri doğru bir sonuç vermezse, else satırında yer alan işlem satırını çalıştırır. else satırı da mevcut değilse, program if-else-if kalıbının hiç bir satırını çalıştırmadan bir sonraki satırdan çalışmasına devam eder. if-else-if kalıbında sadece tek bir satır işlem görür.

Konumuzu açıklamak amacıyla aşağıdaki kodu inceleyebiliriz.

if(i==10){ System.out.println("i nin degeri 10'dur."); } else if(i<10){ System.out.println("i nin degeri 10'dan kücüktür."); } else{ System.out.println("i nin degeri 10'dan farklıdır ve 10'dan kücük degildir."); }

**Sizce yukarıda yazdığımız kodun Console çıktısı nedir ?**

i nin degeri 10'dan farklıdır ve 10'dan kücük degildir.

Örneğin tuz oranı %80 ve üzerinde ise "yüksek derecede tuzlu", %80 ile %50 arasında ise "orta derecede tuzlu", bunların dışında bir durumda ise "düşük derecede tuzlu" şeklinde ekrana bilgiler yazan bir program yazmak istediğimizde if-else if-else yapısını kullanabiliriz.

float saltRatio = 0.9f; if(saltRatio >= 0.8) { System.out.println("yüksek derecede tuzlu"); } else if(0.5 < saltRatio && saltRatio < 0.8 ) { System.out.println("orta derecede tuzlu"); } else { System.out.println("düşük derecede tuzlu"); }

"if-else" yapılarını iç içe de kullanma şansına sahibiz.

Örneğin: 18 yaşından küçük olanlar kan bağışı yapamazlar, fakat, 18 yaşına eşit ve büyük olan bir kişi eğer kilosu 48'den büyükse kan verebilir, kilosu 48'den küçükse kan veremez gibi basit bir kuralı Java'da kodlayalım.

int age=25; int weight=48; if(age>=18){ if(weight>=48){ System.out.println("Kan verebilirsiniz"); } else{ System.out.println("Kan veremezsiniz"); } } else{ System.out.println("Kan verebilmek için yaşınız 18'den büyük olmalıdır."); }

**Sizce yukarıda yazdığımız kodun Console çıktısı nedir ?**

Kan verebilirsiniz.

# Switch-Case Yapısı

Java'da birden çok kod bloğunu koşullara bağlamak için "switch" deyimi kullanılır. Switch Case; tanımlanmış olan yalnızca bir değişkenin, alacağı değerlere bağlı olarak, farklı sonuçlar döndürmesini sağlayan bir yapıdır. Switch kısmında kullanılacak olan değişkenler byte, short, int, char veya String veri tipine sahip olmalıdır.

## Sözdizimi (Syntax)

switch(değer) { case x: // değer x'e eşitse bu kod bloğu çalışacak break; case y: // değer y'ye eşitse bu kod bloğu çalışacak break; default: // değer hiç bir şeye eşit değilse bu kod bloğu çalışacak }

Burada dikkat edilmesi gereken durum switch içine yazdığımız değerleri case deyimi ile kontrol ediyoruz. Switch içinde ki herhangi bir değer caselerden biri ile uyuşuyorsa o case içindeki kod bloğu çalışır.

import java.util.Scanner; public class SwitchCase { public static void main(String[] args) { Scanner input = new Scanner(System.in); int value = input.nextInt(); switch (value) { case 1: // Value 1 break; case 2: // Value 1 break; case 3: // Value 1 break; default: // Value is invalid break; } } }

Yukarıdaki switch-case yapısında hangi kod bloğunun çalıştırılacağını value değişkeni belirler. Value değeri 1 ise case 1, 2 ise case 2, 3 ise case 3 kod bloğu çalıştırılır. Eğer value değişkeninin değeri hiçbir case değeriyle eşleşmiyorsa default alanı çalıştırılır. Her bir case durumunda, switch yapısından çıkmak ve switch'i izleyen bir sonraki ifadeye geçmek için 'break' komutu kullanılır. Break komutunun kullanılmadığı case'lerde ise ilgili case bloğu işletildikten sonra break komutunun bulunduğu satıra kadar olan tüm kod satırları çalıştırılır. Switch-case yapısının çalışma mantığı genel olarak bu şekildedir.

Switch Case ifadeleri, çalışma mantığı açısından baktığımızda If - Else-if ifadeleri ile benzerlik göstermektedir. Ancak çok fazla koşul gerektiren durumlarda If - Else-if ifadelerinin yerine Switch ifadelerini tercih edebiliriz. Bu sayede uzun kod satırlarının aksine daha sade ve düzenli bir görünüm elde edebilir ve kodumuzun okunabilirliğini artırmış oluruz.

Not: "switch-case" yapısında eğer her case'in sonuna "break" ifadesi koymazsak aramış olduğu koşulu bulana kadar tüm case'lere girip o kod bloklarını çalıştıracaktır.

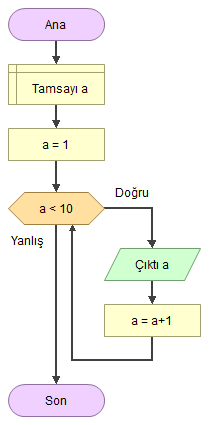
## Örnek

public class SwitchCase { public static void main(String[] args) { int gun = 2; switch (gun) { case 6: System.out.println("Bugün cumartesi"); break; case 7: System.out.println("Bugün pazar"); break; default: System.out.println("Bugün hafta içi"); } // Çıktısı "Bugün hafta içi" } }

# Döngü Yapıları

Döngüler, belirli bir koşul sağlandığı sürece tekrarlanması gereken işler için kullanılan programlama kodlarıdır. Döngü deyimleri, bahsedilen komut parçalarının belirtilen şartlar gerçekleştikçe tekrar tekrar işlenmesini sağlar. Java’da **for, while ve do-while** olmak üzere 3 adet döngü deyimi bulunmaktadır.

Kısacası : Döngü yapıları kodumuzun belirli kısımlarının döngüye girmesini ve birden fazla kez çalışmasını sağlar. Döngüler bir koşula bağlanır ve bu koşul var olduğu sürece aynı kod bloğu çalışmaya devam eder. Koşul artık sağlanmıyorsa döngü sona erer.



# While Döngüsü

**While** döngüsü Java'nın temel yapı taşlarından bir tanesidir. **While** terimini yazdıktan sonra yanına bir parantez açılıp kapatılır ve bu parantezler arasına **boolean** bir değer girilir. Sonrasında bir süslü parantez açılıp kapatılır ve bunun içine kod bloğu yazılır. Bu parantezler içindeki değer **doğru**(true) olduğu sürece döngü dönmeye devam eder ve kod bloğu çalışır.

### While döngüsü söz dizimi şu şekildedir ;

while(kosul){ //kod bloğu }

Burada belirtilen koşul, **boolean** bir ifadedir. Bu koşul **true** olduğu sürece döngü devam eder. Koşul **false** olursa döngü sonlanır. Koşul ifadesi parantez içine yazılır. Eğer döngüye girecek kod tek satırdan oluşuyorsa blok açmaya gerek yoktur; fakat birden fazla satırdan oluşuyorsa mutlaka blok açılmalıdır.

int i = 1; while (i <= 5) { System.out.print(i + ','); i++; }

Bu Java'da **While** kullanımı örneğinde, i değeri 5'e eşit ve küçük olduğu sürece döngü çalışmaya devam edecektir ve çıktısı şu şekilde olacaktır ; 1,2,3,4,5,

Döngülerin içeriği olmak zorunda değildir. Bazı durumlarda döngüye girecek kod olmasa bile döngüye girebiliriz. Aşağıdaki örneği inceleyelim:

int left = 100, right = 200; while (++left < --right); System.out.println("100 ile 200'ün ortası: " + left);

Bu algoritma, 100 ile 200’ün arasında tam ortada bulunan sayıyı bulmamızı sağlar. Kodu çalıştırdığımızda çıktısı şu şekilde olur:

100 ile 200'ün ortası: 150

Gördüğünüz üzere, içeriği olmasa bile bazı durumlarda döngüler fayda sağlayabilir.

Yani while döngüsü genellikle işlemin kaç kez tekrar edeceği bilinmediği durumda kullanılır. Örneğin: müşterinin hesabına giriş şifresini kaç kez yanlış gireceğini bilemeyiz. Bu nedenle bu tarz durumlarda "while" döngüsü tercih edilir hemen aşağıdaki örnekte olduğu gibi.

Scanner scanner = new Scanner(System.in); String customerPassword = "12345"; boolean passwordSuccessfull = false; while(!passwordSuccessfull) { System.out.println("Hesap şifrenizi giriniz:"); String typedPassword = scanner.next(); if(customerPassword.contentEquals(typedPassword)) { passwordSuccessfull = true; System.out.println("Sisteme başarılı giriş yaptınız!"); } }

Java programla dilinde while döngüleri yapacağımız işin ne zaman biteceğiniz bilmediğimiz durumlarda kullanışlı olabilmektedir.

# Do-While Döngüsü

**Do-while** döngüsünün çalışma mantığı **while** döngüsü ile aynıdır , fakat **while** döngüsünde parantez içerisindeki koşul sağlandığı sürece kod bloğu çalışacaktır. Ama **Do-while** döngüsünde durum false olsa bile döngü en az bir kere çalışacaktır. Bu tür durumlar için **Do-while** kullanılır.

Java **Do-While** döngüsü söz dizimi şu şekildedir ;

do { kodlar// } while (kosul);

Gördüğünüz gibi, koşul döngünün sonunda test edilir. Bu durumda, koşul false olsa bile döngü en az bir kere çalıştırılmış olur. Bu şekilde döngü yazmak istediğiniz durumlarla karşılaşacaksınız. Aşağıdaki örneği inceleyelim:

int year = 2020; do { System.out.println(“Döngü işletiliyor..”); year++; } while (year < 2020);

Bu kod çalıştırıldığında çıktısı aşağıdaki gibi olur:

Döngü işletiliyor..

Gördüğünüz gibi year değişkeni 2020’den küçük olmamasına rağmen döngü en az bir kere çalıştırılmıştır.

## Java ****While**** ve ****Do-While**** Arasındaki Farklar ?

Java'da **Do-While** ve **While** döngüleri arasındaki tek fark , **Do-while** döngüsünde, döngü bloğu içindeki kod kesimi **en az bir kez mutlaka** işletilecektir. Çünkü önce döngü bloğu işletilip sonra koşul denetlenmektedir. **While** döngüsünde ise **önce koşula bakılıp** sonra döngü bloğu işletildiği için, döngüye hiç girilmemesi olasıdır.

# For ve While Döngüsü Arasındaki Farklar

Java’da **for, while ve do-while** olmak üzere 3 adet döngü deyimi bulunmaktadır. Birbirinden farklı bu döngülerden hangisini kullanacağız ?

Hiç bir algoritmada hangi döngünün kullanılması gerektiği ile ilgili kesin bir yargı ve kural söz konusu değildir. Bu döngüler uygun şekilde tasarlandıkları sürece aynı işi yapabilirler. Ama genellikle, kaç kere döneceği belli olmayan, koşulun bir girdiye göre denetlendiği durumlarda **While** ya da **do-while** döngüsü kullanılırken, diziler gibi, tekrar sayısı belirli olan durumlarda ise **for** döngüsü kullanılması tercih edilir. Ama yine bu yazılımcının kendisine bağlıdır

# Continue ve Break Komutları

## Continue Deyimi

Java'da "**continue**" deyimi , döngü içinde bir koşul oluştuğunda o döngüyü tamamlamadan bir sonraki kademeye geçmeye yarar.

int i = 0; while (i < 10) { i++; if (i == 5) { continue; } System.out.println(i); }

Örnekte görüldüğü gibi eğer i değişkeni 5 sayısına eşit olduysa, o kod bloğundan sonraki kodlar çalışmayıp döngü bir sonraki adıma geçiş yapacaktır.

## Break Deyimi

Buraya kadar gördüğümüz bütün döngüler belirlediğimiz bir koşula göre kontrol ediliyor ve bu koşul sağlandığı sürece çalışıyordu. Bazı durumlarda, döngü koşulu sağlansa bile başka bir nedenden ötürü döngüyü sonlandırmak isteyebiliriz. Bu tarz durumlarda **break** deyimini kullanırız. Bu deyim, içinde kullanıldığı döngüyü anında sonlandırır.

for (int i = 0; i < 10; i++) { if (i == 5) { break; } System.out.println(i); }

Örnekte görüldüğü üzere döngü içinde ki koşul tamamlanmadan i değişkeni 5 değerine ulaştığında "break" komutu ile döngümüzü bitirebiliriz.

Metotlar (Fonksiyonlar)

Java'da **Metotlar** sadece çağrıldığında kullanılan kod bloklarıdır. Metotlara bir diğer adlandırma olarak **Fonksiyonlar'**da denilmektedir. Bunun sebebi bu kod yapısının matematikte ki fonksiyon mantığına çok benzemesidir. Java'da metotlarımızın içerisine veri aktarmak için ise parametre girebilmekteyiz.

Neden Metot Kullanırız ?

Programlamada metot kullanmamızın sebebi, bir çok yerde kullanacağımız kodu tek seferde yazıp lazım olduğunda çağırmaktır. Örnek vermek gerekirse, programımız içerisinde bir çok yerde matematikteki "üs alma" işlemini gerçekleştirmiş olalım. Her defasında üs alma işlemi için aynı kodları yazmak yerine bu kodları metot olarak yazıp lazım olduğunda çağırmak işlerimizi kolaylaştıracaktır. Ayrıca metotlar "Nesne Yönelimli Programlamanın" yapı taşlarındandır. Programlamada karmaşık bir problemi daha küçük parçalara bölmek, programınızın anlaşılmasını kolaylaştırır ve yeniden kullanılabilir hale getirir.

Metot Tanımlama

Java'da metotların sözdizimi şu şekildedir :

veriTipi metotAdi(parametre1, parametre2, ....) { // kod bloğu }

* veriTipi : Metotlar geriye bir değer döndürebilir, bu değerin veri tipini metot tanımlanırken belirtilir. Örneğin metot geriye "integer" veri tipinde bir değer döndürecekse "veriTipi" kısmına "int" anahtar sözcüğü yazılmalıdır. Eğer metotlar geriye bir değer döndürmeyecekse "void" anahtar sözcüğü kullanılır.
* metotAdi : Metodumuzun benzersiz ismidir ve bu isimlendirme ile metotlar çağrılır.
* kod bloğu : Bu kısım metot çağrıldığı zaman, program tarafından çalışacak kod bloğudur.
* parametre1/parametre2 : Bu kısım metot içerisine aktarma istediğimiz verilerdir ve parametre olarak adlandırılır.

Örneğin :

int toplama(int a,int b) { // kod bloğu }

Yukarıda ki örnekte bir metot tanımladık. Bu metodun adı "toplama" ve geriye döndürdüğü veri tipi "int" türüdür.

Metot Çağırma

Java'da önceden yazdığımız metotları erişimi olduğu yerlerde çağırabiliriz. Java'da bir metodu çağırmak için metodun adını ve ardından iki parantez () ve bir noktalı virgül yazılmalıdır. Aşağıda ki örnekte toplama(); adlı metodun nasıl çağrıldığı hakkında bir örnektir.

public class JavaPatika {static int toplama(int a, int b) { return a + b; } public static void main(String[] args) { int sonuc = toplama(5, 2); System.out.println(sonuc); } } // Çıktısı "7"

# Return ve Void Kullanımı

Java'da iki tür metot mevcuttur bunlar : Return (geri dönüşü olan) , Void (geri dönüşü olmayan) metotlar. Return metotlar çağrıldığında geriye bir değer döndürürken, Void metotlar geriye bir değer döndürmemektedir.

## Return Metotlar

Java'da oluşturduğumuz bir metodun bir değer üretmesini istendiğinde, "return" deyimi kullanılır. Metot içerisinde yazılan kod parçacıkları çalışacaktır, buna ek olarak metot içinde oluşturduğumuz bir veriyi dışarıya aktarma ihtiyacı duyabiliriz, bu durumlarda "return" deyimi kullanılır. Geri dönecek değerin veri tipi, metot tanımlarken verdiğimiz veri tipi ile aynı olmalıdır. Aksi halde derleyici tarafından hata alınır.

public class Main { static int toplam(int x, int y) { return x + y; } public static void main(String[] args) { int t = toplam(5,2); System.out.println(t); } } // Çıktısı "7"

## Void Metotlar

Java'da yazılan bir metodun geriye bir değer döndürmesini istemiyor, sadece metot içindeki kod bloğunun çalışmasını istersek "void" deyimi kullanmalıdır. Metot tanımlanırken veri tipi kısmına "void" deyimi yazılması yeterlidir. Void metotlar içerisinde "return" deyimi kullanılamamaktadır.

public class Main { static void toplam(int x, int y) { System.out.println(x+y); } public static void main(String[] args) { toplam(5,2); } } // Çıktısı "7"

# Metotlarda Overloading (Aşırı Yüklenme)

Java'da, iki veya daha fazla metot, parametreler açısından farklılık gösteriyorsa (farklı sayıda parametre, farklı türde parametre veya her ikisi) aynı isime sahip olabilir. Bu duruma metotlarda "Overloading" yani aşırı yüklenme işlemi denir.

Buradaki amaç aynı işlemi farklı parametrelerle yapacak olan metot ismini tek seferde kullanmaktır. Overloading işlemi "Nesne Yönelimli Programlamada" da çok kullanılmaktadır.

void func() { ... } void func(int a) { ... } float func(double a) { ... } float func(int a, float b) { ... }

Burada func() metodu (overloading) aşırı yüklenmiştir. Bu metotlar aynı isime sahiptir ancak farklı parametreler kabul eder.

Not: Yukarıdaki metotların dönüş türleri aynı değildir. Bunun nedeni, metot aşırı yüklemesinin dönüş türleriyle ilişkili olmamasıdır. Aşırı yüklenmiş metotlar aynı veya farklı dönüş türlerine sahip olabilir, ancak parametreler açısından farklılık göstermeleri gerekir.

public class JavaPatika {static void ekranaYaz(int a) { System.out.println("Parametreler : " + a); } static void ekranaYaz(int a, int b) { System.out.println("Parametreler : " + a + " ve " + b); } public static void main(String[] args) { ekranaYaz(5); ekranaYaz(10, 2); } } // Çıktısı : // Parametreler : 5 // Parametreler : 10 ve 2

# Lokal Değişken Kavramı

Java'da lokal değişkenler, kod blokları içerisinde tanımlanan değişkenlerdir ve sadece tanımlandıkları kod blokları içerisinde kullanılabilirler. Diğer metot ve sınıflar üzerinden erişimleri yoktur. Lokal değişkenler tanımladıkları ({...}) kod blokları arasında kullanılırlar.

public class JavaPatika { static void ekranaBas(){ // Hatalı kullanım

System.out.println(a); }

public static void main(String[] args) {

int a = 10;

if (a < 10) {

int b = 20;

System.out.println(a); } // Hatalı kullanım

System.out.println(b); } }

Bu örnekte if blokları arasında tanımlanmış olan integer türünde ki "b" değişkeni, if bloğunun dışında ekrana bastırılmak istenildiğinde hata ile karşılaşılacaktır. Çünkü "b" değişkeni if blokları arasında tanımlanmıştır ve tanımlandığı kod blokları arasında kullanılabilir.

Dikkat edilmesi gereken diğer bir nokta, main metodun içerisinde tanımlanmış olan "a" değişkeni if bloğu içerisinde kullanılabilir, çünkü if bloğu main metodun bir parçasıdır. Ama "a" değişkeni ekranaBas(); metodu içerisinde kullanılamaz, bunun sebebi ise ekranaBas() ve main metodu farklı iki kod bloklarıdır.

# Recursive (Özyineli) Metotlar

Java'da **Recursive Metotlar**, bir metodun kendisini çağırma tekniğidir. Bu teknik karmaşık problemleri ,çözmesi daha kolay problemlere ayırmayı sağlar. İki sayıyı birbirine eklemek kolaydır, ancak bir dizi sayıyı birbirine eklemek daha karmaşıktır. Recursive metotlar sürekli kendilerini çağırdıkları için dikkat edilmesi gereken durum en son aşama için koşul koyulmasıdır .Aşağıdaki örnekte recursive ile bir dizi sayıyı , iki sayı ekleme basit görevine bölerek toplamak için kullanılır:

public class Main { public static void main(String[] args) { int sonuc = toplam(10); System.out.println(sonuc ); } public static int toplam(int k) { if (k > 0) { return k + toplam(k - 1); } else { return 0; } } }

toplam() metodu çağrıldığında, k'dan küçük tüm sayıların toplamına k parametresini ekler ve sonucu döndürür. K değişkeni 0 olduğunda, metot sadece 0 döndürür. Metot çalışırken, program şu adımları izler:

f(1) = 1;

f(2) = f(1) + 2;

f(3) = f(2) + 3;

f(4) = f(3) + 4;

..........

..........

f(n) = f(n-1) + n;

Sınıf ve Nesne Kavramları

Nesne (Object) Nedir ?

Nesne tanımının TDK'da ki tanımına bir bakalım :

**1.** *isim* Belli bir ağırlığı ve hacmi, rengi olan her türlü cansız varlık, şey, obje.

**2.** *isim, dil bilgisi* Geçişli fiili bütünleyen yalın veya belirtme durumunda bulunan tümleç: *Ali bir kitap almış cümlesinde kitap nesnedir.*

**3**. isim, felsefe Öznenin dışında kalan her konu, obje: "*Her nesne ve olaya alaycı bir gözle bakmak ilkesinden yola çıkar bu görüş.*" - **Salâh Birsel**

**3.** *isim, felsefe* Öznenin dışında kalan her konu, obje: "*Her nesne ve olaya alaycı bir gözle bakmak ilkesinden yola çıkar bu görüş." -* **Salâh Birsel**

Nesnelere Örnekler

* Araba, telefon, elma, ördek, kalem gibi günlük hayatta kullandığımız eşyalar.



Bildiğimiz gibi her nesnenin kendine ait **nitelikleri** ve **davranışları** vardır. Nesneler birbirlerinden farklıdır ve kendi **varoluşlarına** göre davranırlar ve kendi kimliklerine sahiptirler.

**Nitelik** : Nitelik kavramı, bir nesnenin özellikleridir ve nesnenin mevcut durumunu tanımlar. Mesela bir ördeğin rengi ve ağırlığı o ördeğin nitelikleridir. Bir ördeğimiz rengi siyah, diğer ördeğimizin rengi beyazdır. Bu durumda her iki nesne (yani ördek) nitelikleri sebebi ile birbirilerinden bağımsızdırlar.

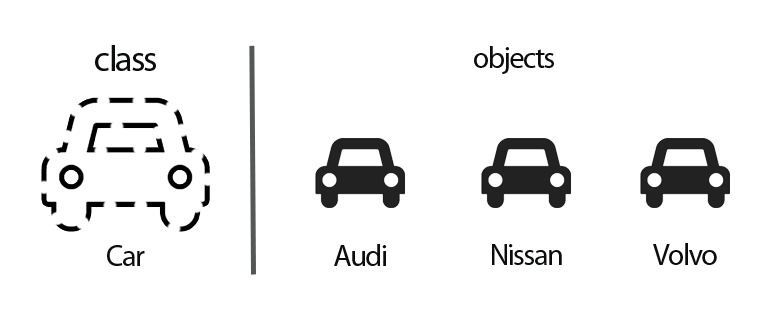
**Davranış** : Bir nesnenin kendine özel yaptığı eylemlerdir. Yine ördek örneğine bakarsak, bir ördek uçabiliyorken diğer bir ördek fiziksel durumu gereği uçamayabilir.

Nesneler hakkında bilmemiz gereken bir diğer husus ise, nesneler bir isimdir. Nesnelerin kendilerine ait nitelik ve davranışları vardır.

Sınıf (Class) Nedir ?

NYP sınıflar ve nesneler üzerine kurulmuştur, "**Sınıflar**" bir problemi soyutlamak ve genelleştirmek için kullanılan yapılardır veya kılavuzlardır. Sınıflar, bir nesneye ait tüm özellikleri temsil eder. Bu özellikler nesnenin ne tür **nitelikleri ve davranışları** olacağını belirler.

Mesela "**Araba**" bir sınıftır. Arabalara ait nitelikler **renk, hız, vites sayısı, yakıt türü vb.** bir sürü nitelik olabilir. Ayrıca bazı arabalara özel davranışlar olabilir, park sensörü, oto pilot, hız sabitleme gibi arabaların kendilerine özel davranışları da olabilir. Bir araba üretilirken, bir yapım kılavuzuna ihtiyaç vardır. Programlama da bu kılavuzlara "**Sınıf (Class)**" denir.



# Sınıflar (Classes)

Java Nesne Yönelimli bir programlama dilidir. Java'daki her şey, değişkenleri ve metotları ile birlikte sınıflar ve nesnelerle ilişkilidir. Örneğin: gerçek hayatta araba bir nesnedir. Otomobilin ağırlık ve renk gibi değişkenleri ve sürüş ve fren gibi metotları vardır. Nesne yönelimli programlamanın amacı yazdığımız kodlara soyut bir kavrama dönüştürmektir.

Sınıflara ait nitelikler ve davranışlar vardır. Programlamada nitelikler için değişkenler (variable) ,davranışlar için ise metotlar (method) tanımlanır.

## Sözdizimi (Syntax)

class <class\_name> { <variable>; <methods>; }

Java'da "**sınıf**" tanımlanırken "**class**" deyimi kullanılır ve sonrasında sınıf ismi yazılır. Java'da sınıf isimleri her zaman büyük harf ile başlaması gerekmektedir ve oluşturulan Java dosyası ile aynı isimde olmalıdır.

Sınıf ismini de verdikten sonra "{" işareti ile sınıfa ait kapsamı yani kod bloğunu oluştururuz. "}" parantezi ile de sınıfa ait kapsamı kapatırız. Böylece, sınıfımız için yazacağımız kodlar "{}" arasında kalan alanda yazılacaktır. Bu da sınıfın kapsamını ifade eder.

Sınıf kod bloğunu açtıktan sonra bu kod bloğu için değişkenleri ve metotları yazarız. Unutmayınız ki metotlar da kendilerine ait kod blokları, yani kapsamları vardır. Onları da "{}" ile belirtiriz.

class Car { // nitelikler String type; String model; String color; int speed; // davranışlar int increaseSpeed(int increment) { speed += increment; return speed; } int decreaseSpeed(int decrease) { if (speed > 0) { speed -= decrease; } return speed; } void printSpeed() { System.out.println("Speed : " + speed); } // ... }

Yukarıda sınıf ve fonksiyon tanımlamalarına detaylıca yer verilmiştir.

Sınıflar nesneler oluşturabilmek için yazılım dünyasında oluşturulmuş şablonladır. Bu şablon nesne ile ilgili modellenecek tüm özellikleri ve davranışları bir taslak halinde kodlanmasını sağlar. Böylece, tanımlanmış bir sınıftan binlerce nesne oluşturabiliriz.

# Nesne Oluşturma

Sınıflar nesneleri tarif eden şablonlardı. Nesneler ise bu şablonlardan üretilen fiziksel yapılardır. Her üretilen nesne Hesap Hafıza Bölgesi'nde tutulur. Böylece sınıftan fiziksel karşılığı olan bir yapı elde etmiş oluruz. Sınıftan onlarca, yüzlerce nesne yaratabiliriz. Hepsi de hafıza başka adresleri gösterirler.

public class Car { // nitelikler String type; String model; String color; int speed; // davranışlar int increaseSpeed(int increment) { speed += increment; return speed; } int decreaseSpeed(int decrease) { if (speed > 0) { speed -= decrease; } return speed; } void printSpeed() { System.out.println("Speed : " + speed); } // ... }

Java'da nesne üretmek için kullanılan sözdizimi :

ClassName object = new ClassName();

* **ClassName** : Nesne oluşturmak istediğimiz sınıfı belirtiyoruz. Bu sınıf daha öncesinde projemizde tanımlanmış olması gerekmektedir.
* **object** : Nesnemize verdiğimiz isimdir ve aynı isimde birden fazla nesne oluşturulamaz.
* **new** : Java'da nesne üretmek için "**new**" anahtar kelimesini kullanırız.
* **ClassName();** : Sınıfa ait Kurucu (Constructor) Metodu temsil eder. Varsayılan olarak parametresiz tanımlanır.

Car sınıfına ait örnek nesne oluşturma :

Car audi = new Car(); Car bmw = new Car(); Car mercedes = new Car();

Yukarıdaki örnekte "Car" sınıfına ait 3 tane farklı nesne ürettik. Bu nesnelerin her birinin nitelikleri farklı olmakla beraber hafızada ayrı ayrı yer kaplamaktadırlar.

Sınıf Niteliklerine Erişim

Sınıflara ait niteliklere erişim sağlamak için nokta (.) kullanılır. İlgili nesnenin ismini sonuna nokta koyularak erişilmek istenilen niteliğin ismi yazılır.

public class Main { public static void main(String[] args) { Car audi = new Car(); audi.speed = 10; System.out.println("Audi Hızı : " + audi.speed); Car bmw = new Car(); bmw.speed = 25; System.out.println("Bmw Hızı : " + bmw.speed); Car mercedes = new Car(); mercedes.speed = 30; System.out.println("Mercedes Hızı : " + mercedes.speed); } }

Programın çıktısı :

Audi Hızı : 10 Bmw Hızı : 25 Mercedes Hızı : 30

Sınıf Metotlarına Erişim

Sınıfa ait davranışlara yani metotlara erişmek için nokta (.) kullanılır. İlgili nesnenin ismini sonuna nokta koyularak erişilmek istenilen metodun ismi yazılır ve var ise parametreleri girilir.

public class Main { public static void main(String[] args) { Car audi = new Car(); audi.speed = 10; audi.increaseSpeed(20); audi.printSpeed(); Car bmw = new Car(); bmw.increaseSpeed(10); bmw.increaseSpeed(25); bmw.increaseSpeed(5); bmw.decreaseSpeed(25); bmw.printSpeed(); Car mercedes = new Car(); mercedes.speed = 20; mercedes.printSpeed(); } }

Programın çıktısı :

Speed : 30 Speed : 15 Speed : 20

Constructor (Kurucu) Metotlar

Kurucu metotlar sınıf tasarlanırken yazılırlar. Sınıfınızı yazarken kurucu metotlarınızı da tanımlayabilirsiniz. Eğer sınıf içinde hiç kurucu metot tanımlamazsınız parametresiz boş bir kurucu metot Java tarafından otomatik olarak tanımlanır.

Kurucu metotlar ilgili sınıftan bir nesne üretmeye çalıştığınızda daha nesne üretme aşamasında çalıştırılan özel metotlardır (fonksiyonlardır). **Kurucu metotların isimleri Sınıf ismiyle aynı olmak zorundadır.** Dönüş tipi olarak veya void olarak herhangi bir tanımlama yapılmasına gerek yoktur.

"**new**" anahtar kelimesi ile nesne üretirken kurucu metot çağrımı yapılır. İki tip kurucu metot vardır:

* Parametresiz Varsayılan Kurucu Metot
* Parametreli Kurucu Metot

Car sınıfımız :

public class Car { // nitelikler String type; String model; String color; int speed; // Constructor (Kurucu) Metot Car(String type, String model, String color) { this.type = type; this.model = model; this.color = color; this.speed = 0; } // davranışlar int increaseSpeed(int increment) { speed += increment; return speed; } int decreaseSpeed(int decrease) { if (speed > 0) { speed -= decrease; } return speed; } void printSpeed() { System.out.println("Speed : " + speed); } void printInfo() { System.out.println("================"); System.out.println("Model : " + this.model); System.out.println("Color : " + this.color); System.out.println("Type : " + this.type); System.out.println("Speed : " + this.speed); } // ... }

Nesnelerimiz :

public class Main { public static void main(String[] args) { Car audi = new Car("Sports", "Audi", "red"); audi.increaseSpeed(20); audi.printInfo(); Car bmw = new Car("Sports" , "BMW" , "blue"); bmw.increaseSpeed(10); bmw.increaseSpeed(25); bmw.increaseSpeed(5); bmw.decreaseSpeed(25); bmw.printInfo(); Car mercedes = new Car("Corporate" , "Mercedes" , "black"); mercedes.speed = 20; mercedes.printInfo(); } }

Programın çıktısı :

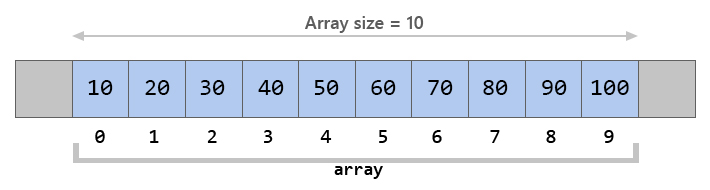
================ Model : Audi Color : red Type : Sports Speed : 20 ================ Model : BMW Color : blue Type : Sports Speed : 15 ================ Model : Mercedes Color : black Type : Corporate Speed : 20

# Dizilerin Genel Mantıkları

Java'da Diziler (Arrays) , her değer için ayrı değişkenler oluşturmak yerine, birden çok değeri tek bir değişkende depolamak için kullanılır. Değişkenler hafızada tek bir değer tutmamızı sağlar. Fakat bazı durumlarda, birden çok veriyi bir arada bulundurmak isteriz. Örneğin, bir sınıfta okuyan 20 öğrenci olsun. Bu 20 öğrenci için hafızada ayrı ayrı 20 tane değişken oluşturmak yerine, tek bir değişken kullanarak 20 öğrencinin koleksiyonunu tutmak isteyebiliriz.

Bu gibi durumlarda dizileri kullanırız. Dizi, aynı türden birden fazla değişkeni tutmamızı sağlayan hafıza birimidir. Kısaca, dizileri aynı türden elemanları gruplamak için kullanırız. Dizi oluşturduktan sonra dizinin içerisindeki elemanlara indeks numarasıyla ulaşır ve değiştiririz. Ayrıca, çok boyutlu diziler oluşturmak da mümkündür. Tek boyutlu dizi kullanabileceğimiz gibi 2 veya 3 boyutlu diziler de oluşturabiliriz. 2 boyutlu dizilere matris denir.

Dizi (Array) kavramı programlama dillerinde bir veri tipini ifade eder. Bu veri tipi liste halindeki ardışık verileri bir arada tutan yapıya denilir. Bu ardışık yapıya ait elemanlara indis yoluyla erişim sağlanabilir. Diziler sabit boyutludur ve tanımlanırken boyutları belirtilir.



Yukarıdaki örnek 10 eleman boyutunda "Integer" veri tipi tutan bir dizidir. Dizilerdeki herhangi bir elemana ulaşmak için indis numaraları kullanılır. Dizi indisleri her zaman 0'dan başlar o yüzden dizinin son elemanının değeri, dizi boyutuna eşit değildir ve bu dikkat edilmesi gereken bir konudur.

Dizinin hafızada bir başlangıç adresi olur ve ardışık olan diğer elemanlar sırayla hafızaya yerleştirilir. Diziler "new" anahtar sözcüğüyle oluşturulur. Böylece, Heap Hafıza bölgesinde yer kaplarlar.

double[] myList; // tercih edilen yol veya double myList[]; // başka türlü tanımlama biçimi

Diziler veri tipi ve [] parantezler ile belirtilir. Yukarıda iki farklı tanımı görülmektedir. Hafızadan yer alıp diziye alan ayırabilmek için "new" anahtar kelimesi kullanılır.

double[] myList = new double[10];

Yukarıda maksimum 10 eleman alabilen "double" veri tipinde olan bir dizi oluşturulmuştur.

Yukarıda "myList" isminde bir dizi tanımlamıştık. Tanımlanan dizi hafıza yukarıdaki şekildeki gibi tutulur. "myList" değişkeni dizinin başlangıç adresini tutar. Dizilerde ardışık bir sıra olduğu için ilk elemandan sonra gelen elemanların hafıza adresleri de birer birer artar. Dizi blok halinde yer kaplar. Diziye erişmek için indeks numarası kullanılır. Örneğin: "myList[0]" demek dizinin 1. Elemanını verecektir. Java'da dizilerin indeksleri sıfırdan başlar. Örneğin: "myList[5]", yani 5 nolu indeksteki dizi elemanını ver dediğimizde aslında dizinin 6. Elemanına erişmiş oluruz.

Örnekler:

// Java'da diziye ilk değerler süslü parantezler arasında verilir. double[] myList = { 1.9, 2.9, 3.4, 3.5 }; // tüm dizi elemanlarını arada boşuk bırakarak sırayla ekrana yazdırır. for (int i = 0; i < myList.length; i++) { System.out.println(myList[i] + " "); }

Yukarıdaki örnekte myList isimli diziye ilk değerleri hemen verilmiştir. Süslü parantezler arasında kalan değerler dizi elemanlarıdır.

Ardından, bir "for" döngüsü ile dizi elemanları ekrana yazdırılır.

### Dizileri Fonksiyonlara Parametre Olarak Göndermek

Tanımladığınız dizileri fonksiyonlara parametre olarak gönderebilirsiniz.

Örneğin elimizde static bir fonksiyon olsun. Bu fonksiyona int tipinde verileri olan bir diziyi girdi (input) olarak vermek istersek aşağıdaki gibi olur.

public static void printArray(int[] array) { for (int i = 0; i < array.length; i++) { System.out.print(array[i] + " "); } }

printArray ( int[] array ) kırmızı olarak işaretlediğimiz yer diziyi yerel değişken olarak fonksiyona gönderdiğimiz noktadır. Java'da tüm değişkenler "Pass by Value" yöntemiyle geçilir. Bu şu demektir. Sizin tanımladığınız değişkenin, nesnenin veya dizinin birebir kopyası oluşturulur. Bu kopya değer fonksiyona yerel değişken olarak gider. Bu Java mülakatlarında size sorulabilecek bir detaydır.

### Dizileri Fonksiyonlardan Geri Döndürmek

Fonksiyonlar belli bir işi yapıp sonucunda değer dönebilen veya dönmeyen kod bloklarıdır. Fonksiyonlar için altın kural, her fonksiyonun tek bir işi olmalıdır. Örneğin: dizi ortalaması alma işi yapan bir fonksiyon dizileri ekrana yazdırma işini yapmamalıdır. Veya dizilerin ortalamasını alma işi ile dizileri toplama işlemi aynı fonksiyon içinde olmamalıdır. Her biri ayrı fonksiyonlar olmalıdır.

public static int[] reverse(int[] list) { int[] result = new int[list.length]; for (int i = 0, j = result.length - 1; i < list.length; i++, j--) { result[j] = list[i]; } return result; }

Yukarıda dizinin tersine çevrilmiş halini döndüren bir fonksiyon vardır. public static **int[]** reverse(…) koyu renkle işaretlenen alan dizi döndüreceğimizi ve bu dizinin veri tipini söylüyoruz. Burada veri tipimiz "int".

# Tek Boyutlu Diziler

Tek boyutlu diziler basitçe, aynı türden elemanların listesini tutan bir yapıdır. Dizi oluşturmak için, önce dizide yer alacak elemanların türü belirtilir, sonra diziye bir isim verilir ve isimden sonra köşeli parantezler ( [ ve ] ) konulur.

int numbers[]; // Burada numbers isminde bir dizi oluşturuluyor

Köşeli parantezleri değişken isminden sonra koymak yerine, tür isminden sonra da yazabilirsiniz. Örneğin aşağıdaki kodun yukarıdakiyle bir farkı yoktur:

int[] numbers; // Burada numbers isminde bir dizi oluşturuluyor

Diziler new deyimiyle oluşturulur. Dizi oluştururken kapasite değeri vermek **zorunludur**. Kapasite değeri, dizinin kaç eleman barındıracağını belirtir. Aşağıdaki örnekte, 5 adet int değişkeni tutabilecek bir dizi oluşturuluyor:

int[] numbers = new int[5];

Bu ifade çalıştırıldığında, hafızada 5 adet int değişken için yetecek kadar alan ayrılır. Bu alanı düzenleyebilmek için indeks numaraları kullanırız. **Dizi indeksleri 0’dan başlar** ve kapasitenin 1 eksiğine kadar gider. Örneğin, yukarıdaki dizinin indeksleri 0’dan 4’e kadardır. Şimdi bu dizinin ilk elemanını verelim:

numbers[0] = 10; // Dizinin ilk elemanı 10 olarak ayarlandı.

Dizinin diğer elemanlarını şu şekilde verelim:

numbers[1] = 15; numbers[2] = 20; numbers[3] = 25; numbers[4] = 30;

Bu kodlar çalıştırıldığında dizinin elemanları sırasıyla aşağıdaki gibi olur:

{ 10, 15, 20, 25, 30 }

Aşağıdaki kodu çalıştırdığınızda konsola 25 yazar:

System.out.println(numbers[3]);

Dizilerle uğraşırken indeks numaralarına çok dikkat etmelisiniz. Eğer dizinin aralığı dışında bir indekse erişmeye çalışırsanız, \***IndexOutOfBoundsException\*** hatası meydana gelir.

System.out.println(numbers[5]); // Hata!

Yukarıdaki satır hataya neden olur; çünkü numbers dizisinin kapasitesi 5 olmasına rağmen dizinin 6. elemanına erişmeye çalışıyoruz.

Eğer dizinin içindeki elemanlar dizi oluşturulurken belliyse, diziyi oluştururken elemanları küme parantezi içinde ve virgülle birbirinden ayırarak verebiliriz:

String[] weekDays = new String[] { "Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma", "Cumartesi", "Pazar" };

Bu şekilde oluşturulan dizilere kapasite vermemize gerek yoktur; çünkü kapasite değeri zaten eleman sayısından bellidir. Yukarıdaki örnekte **weekDays** dizisinin kapasitesi otomatik olarak 7 olur.

Yukarıdaki gibi dizi oluştururken new deyimini kullanmaya gerek yoktur. Yani, yukarıdaki kodu aşağıdaki gibi yazabiliriz:

String[] weekDays = { "Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma", "Cumartesi", "Pazar" };

### Dizinin Kapasitesini Öğrenmek

Her dizinin **length** adında bir özelliği bulunur. Bu özelliği kullanarak dizinin kapasitesini öğrenebilirsiniz.

Örneğin, aşağıdaki kodu inceleyelim:

int[] numbers = new int[100]; System.out.println(numbers.length); // Konsolda 100 yazar

### Dizinin Bir Elemanını Değiştirme

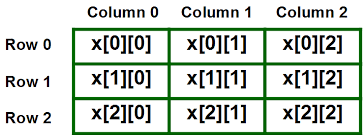
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"}; cars[0] = "Opel"; cars[2] = "Toyota"; System.out.println(cars[0]);

# Çok Boyutlu Diziler

Java'da Çok Boyutlu Diziler varsayılan bir veri tipi olarak bulunmazlar ve matris olarak adlandırılırlar. Dizilerin 2 boyutlu halleri şeklinde tanımlanırlar. Matrisler satır ve sütun şeklinde tablo verisi formatındaki verileri tutmak için kullanılır. Diziler liste halinde veriler için uygunken, matrisler tablo şeklindeki veriler için uygundur. Oluşturulan tabloda bir değere ulaşmak istersek satır ve sütun sayısını girmemiz yeterli olacaktır.

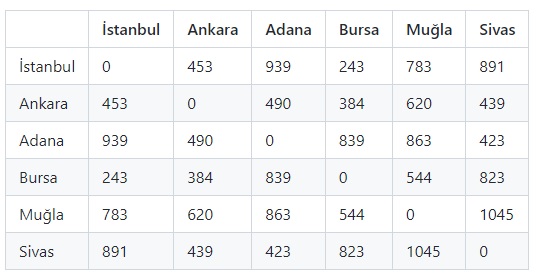
İki boyutlu dizilerde tek boyutlu diziler gibi indis değeri 0'dan başlar. Dizide tutulacak veri tipleri aynı olmak zorundadır farklı veri tiplerini aynı matriste tutamayız.

3x3 boyutunda bir matris örneği :



Tablo oluşturup bu tablonun bilgilerini matris ile gösterelim. Örneğin, şehirler arasındaki mesafeyi gösteren bu tabloyu uzaklık değişkenimizde sakladık.

Mesafe Tablosu(KM)



int[][] uzaklik ={ {0, 453, 939, 243, 783, 891}, {453, 0, 490, 384, 620, 439}, {939, 490, 0, 839, 863, 423}, {243, 384, 839, 0, 544, 823}, {783, 620, 863, 544, 0, 1045}, {891, 439, 423, 823, 1045, 0} };

##### **İki Boyutlu Dizi Tanımlama Yöntemleri**

İki boyutlu dizileri oluşturmanın farklı yöntemleri var şimdi bunlara bakalım.

Bir değişkenin dizi olduğunu köşeli parantezler ile belirtmiştik. Bir tane köşeli parantez tek boyutlu dizi belirtir. Eğer çok boyutlu dizi oluşturmak istiyorsak, boyut sayısı kadar köşeli parantez belirtmeliyiz. Örneğin aşağıdaki satır 2 boyutlu bir dizi (yani matris) belirtir:

degiskenTipi[][] arrayIsmi; int matrix[][];

yada

degiskenTipi arrayIsmi[][]; //bu yöntem tercih edilmez

İki boyutlu dizimizi tanımlarken değerlerini atamak istersek aşağıdaki yöntemi kullanabiliriz.

int[][] array = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12} };

İlk köşeli parantez birinci boyutu (satırları), diğeri ise ikinci boyutu (sütunları) belirtir. Aşağıdaki kodu çalıştırırsak, 3 satırlı ve 4 sütunlu bir matris oluşturur:

int matrix[][] = new int[3][4];

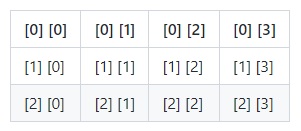
yada

matrix = new int[3][4];

oluşturulan matrix değişkeninin default değeri (a) ile gösterilen tablodaki gibidir.

[0][1][2][3] [0] 0 0 0  [1] 0 0 0  [2] 0 0 0  (a)

Bu matrisin bütün elemanlarına ulaşmak için kullanmamız gereken indeks numaralarını aşağıdaki tabloda görebilirsiniz:



matrix[1][2]; // Matrisin 2. satır ve 3. sütunundaki elemana erişiliyor matrix[0][3]; // Matrisin 1. satır ve 4. sütunundaki elemana erişiliyor matrix[2][0]; // Matrisin 3. satır ve 1. sütunundaki elemana erişiliyor

Matrisin 2. satırı ve 3. sütununda yer alan değerini 7 yapalım.

matrix[2][1] = 7;

Atadığımız 7 değerine ulaşmak istersek matrix[2] [][][1] ifadesini kullanırız.

Fark ettiğiniz gibi matrislerin dizilerden tek farkı [] parantez yanına bir tane [] açıyoruz. Zaten her [] ifadesi yeni bir boyut anlamına geliyor. "**double[][][] ucBoyutlu;**" şeklinde bir ifadeyle 3 boyutlu veri saklayan bir veri yapısı oluşturmuş oluyoruz.

Şimdi güzel bir örnek yapalım. 3 satırdan ve 4 sütundan oluşan bir matris yaratalım ve bu matrisin elemanlarını sırayla 1’den başlayacak şekilde dolduralım. Aşağıdaki kodu inceleyelim:

int[][] matrix = new int[3][4]; int number = 1; for (int x = 0; x < matrix.length; x++) { int[] row = matrix[x]; for (int y = 0; y < row.length; y++) { row[y] = number; number++; } }

Şimdi yukarıdaki kodu inceleyelim. İki boyutlu diziyi oluşturduktan sonra önce for döngüsüyle dizinin satırlarını geziyoruz. Daha sonra içerideki for döngüsüyle dizinin sütunlarını dolaşıyoruz. Bu örneği vermemizdeki amaç, matrisin elemanlarına ulaşmak için iç içe 2 for döngüsü kullanmak gerektiğini göstermektir. Ayrıca dizinin length metodunun faydasını da burada görmüş oluyoruz.

Yukarıdaki kod çalıştığında matrisin elemanları şu şekilde olur:

##### 

**İç İçe Döngülerle Dizilere Erişim ve Dizi İşlemleri**

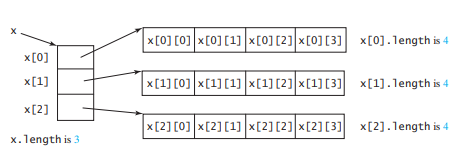
Aşağıdaki kod bize bi matris değerini ekrana nasıl yazdıracağımızı gösterir. Döngüye girdikten sonra her sütun sayısı her satır için tek tek ekrana bastırılır. Her bir satırın tamamlanmasından sonra System.out.println() ile bir alt satıra geçilir.

for (int row = 0; row < matrix.length; row++) { for (int column = 0; column < matrix[row].length; column++) { System.out.print(matrix[row][column] + " "); } System.out.println(); }

##### **İki Boyutlu Dizilerin Uzunlukları**

İki boyutlu diziler aslında her satırının tek boyutlu olduğu dizilerdir. Çok boyutlu dizilerinde boyutuna ulaşarak işlem yapabiliriz.

Örneğin x = new int[3] [][][4] arrayinin x[0], x[1], x[2] değerleri tek boyutlu dizidir. Ve her biri aşağıdaki şekilde gösterildiği gibi 4 elemanı vardır. x.length değeri 3'tür, yani bize sahip olduğu 3 satırın değerini verir. Sütun sayısına ise her hangi bir satır değerinin uzunluğu kadardır. x[2].length değeri 4'tür.

[](https://github.com/Kodluyoruz/taskforce/blob/main/java/java-101/matris-islemleri/figures/matrices_5.PNG)

Yukarıda öğrendiğimiz konuları pekiştirmek adına aşağıdaki kodu inceleyelim.

public double[][] multiplyMatrices(double[][] firstMatrix, double[][] secondMatrix) { // firstMatrix.length ile ilk matrisin satır sayısını buluyoruz. örneğimizde 3 olarak gelecektir. // secondMatrix[0].length ile ikinci matrisin sütun sayısını buluyoruz. örneğimizde 4 olarak gelecektir. // C matrisi olacak olan matrisi tanımlıyoruz. double[][] result = new double[firstMatrix.length][secondMatrix[0].length]; // matrislerde çarpma, toplama gibi işlemleri yapabilmek için iç içe 3 tane döngüye ihtiyaç duyarız. for (int row = 0; row < result.length; row++) { for (int col = 0; col < result[row].length; col++) { result[row][col] = multiplyMatricesCell(firstMatrix, secondMatrix, row, col); } } return result; } double multiplyMatricesCell(double[][] firstMatrix, double[][] secondMatrix, int row, int col) { // A matrisinin satırı ile B matrisinin sütunu çarpma işlemi. /\* \* row = 3, col = 4 olarak gelecektir. \* i = 0 ise \* firstMatrix[3][0] \* secondMatrix[0][4] \* i = 1 ise \* firstMatrix[3][1] \* secondMatrix[1][4] \* i = 2 ise \* firstMatrix[3][2] \* secondMatrix[2][4] \* şeklinde i değişerek satır ve sütun çarpılır. \*/ double cell = 0; for (int i = 0; i < secondMatrix.length; i++) { cell += firstMatrix[row][i] \* secondMatrix[i][col]; } return cell; }

Yukarıdaki “multiplyMatrices” fonksiyonu iki tane matrisi girdi olarak alır. Fonksiyon matrislerin çarpım sonucunu matris olarak döndürür.

“multiplyMatrices” fonksiyonu içinde bir başka fonksiyon daha çağrılmıştır. “multiplyMatricesCell” bu fonksiyon ise iki matrisin satır ve sütununu çarpıp sonucu double tipte bir değer döndürür.

## Sütun Kapasiteleri Farklı Matris Oluşturmak - Düzensiz Diziler

Farklı sütun değerine sahip dizilere "**Ragged Arrays**" yani düzensiz diziler denir. Başka bir açıdan bakıldığında ise, 2 boyutlu dizileri, **dizilerin dizisi** (**array of arrays**) olarak düşünmek doğru olur. Yani iki boyutlu dizileri tek boyutlu diziler oluşturur.

Diziler aynı türden elemanlardan oluşmak zorundadır. int türünde bir dizi olabileceği gibi, dizinin dizisi de olabilir.

Yukarıdaki örneklerde matrisin sütun sayısını 4 olarak belirledik. Bu şekilde oluşturulursa matrisin bütün satırları 4 elemanlı olur. Fakat bu zorunlu değildir. Matris oluştururken sütun sayısı belirlemezsek, her bir satırdaki dizilerin kapasitesi farklı olabilir. Örneğin aşağıdaki kodu inceleyelim:

int[][] matrix = new int[3][]; matrix[0] = new int[1]; matrix[1] = new int[2]; matrix[2] = new int[3];

Burada önce 3 satırdan oluşan bir matris belirttik, fakat sabit bir sütun sayısı vermedik. Sonra her bir satır için ayrı ayrı sütun sayısı belirledik.

Çok boyutlu dizi oluştururken, yalnızca ilk boyutun (en soldaki) kapasitesini belirlemeniz yeterlidir. Diğer boyutların kapasitesini dinamik olarak belirleyebilirsiniz.

#### **Sıra Sizde!**

Aşağıda verilen örneklerin cevaplarına bakmadan önce kendiniz kodlamaya çalışmanızı öneririz. Daha sonra zorlandığınız noktalarda ufak yardımlar almak için bakabilirsiniz. Bu sizin öğrenmenize daha yardımcı olacaktır.

Matris boyutunu sizin belirlediğiniz matris değerlerini ise kullanıcının girdiği bir algoritma yazınız.

Cevap:

int matrix[][] = new int[3][4]; Scanner input = new Scanner(System.in); System.out.println("Enter " + matrix.length + " rows and " + matrix[0].length + " columns: "); for (int row = 0; row < matrix.length; row++) { for (int column = 0; column < matrix[row].length; column++) { matrix[row][column] = input.nextInt(); } }

İstediğiniz boyutta matris oluşturarak matris değerlerine 0-99 arası random sayılar üreterek matrise atama yapınız.

Cevap:

int[][] matrix = new int[3][4]; for (int row = 0; row < matrix.length; row++) { for (int column = 0; column < matrix[row].length; column++) { matrix[row][column] = (int) (Math.random() \* 100); } }

Aşağıdaki kod bloğunun çıktısı nedir?

int[][] array = {{1, 2}, {3, 4}, {5, 6}}; for (int i = array.length - 1; i >= 0; i--) { for (int j = array[i].length - 1; j >= 0; j--) { System.out.print(array[i][j] + " "); } System.out.println(); }

Cevap:

6 5 4 3 2 1

# ForEach Kullanımı

Java'da ForEach deyimi adında da anlaşılacağı gibi bir döngüyü ifade eder. Genelde dizilerdeki ve listelerdeki elemanları daha hızlı şekilde ulaşmak için kullanılan kısa bir yöntemdir. Kısacası for döngüsünün modifiye edilmiş halidir.

for (veritipi degisken: diziAdi) { // kod bloğu }

Örnekte Java'da ForEach kullanımını görmektesiniz, yine bir for döngüsü oluşturuyoruz ama iki parametreyi iki nokta ile ayırıyoruz. İlk kısmında dizinin döndüreceği veri tipini ve for içinde dizinin elemanı olarak kullanacağınız değişkeni tanımlıyorsunuz, diğer kısımda ise dizimizin ismini yazıyoruz.

Bu örneğimizde arabalar dizisinin forEach ile nasıl kullanıldığını görmektesiniz.

String[] arabalar = {"BMW", "Mercedes", "Ford", "Ferrari"}; for (String i : arabalar) { System.out.println(i); } // Çıktısı // BMW // Mercedes // Ford // Ferrari

Çok boyutlu dizilerde forEach kullanımı

public class Main {public static void main(String[] args) { int[][] matris = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12} }; for (int[] u : matris) { for (int elem : u) { System.out.println(elem); } } } }

# Arrays Sınıfı ve Metotları

Java.util paketindeki Arrays sınıfı, Java Collection Framework'ün bir parçasıdır. Bu sınıf, Java dizilerini dinamik olarak oluşturmak ve bunlara erişmek için statik metotlar sağlar. Yalnızca statik metotlar ve Object sınıfının metotlarından oluşur. Bu sınıfın metotları, sınıf adının kendisi tarafından kullanılabilir.

import java.util.Arrays;

Import deyimi ile java.util.Arrays sınıfını projeye dahil etmemiz gerekmektedir.

## Arrays.toString()

Diziye ait elemanları direk ekrana basmak için kullanılan bir metottur.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] dizi = {3, 5, 79, 12, 25, -3, 66, 82, -49, 152}; System.out.println(Arrays.toString(dizi)); } } // Çıktısı // [3, 5, 79, 12, 25, -3, 66, 82,-49,152]

## Arrays.fill()

Arrays.fill metodu ile dizilerimizin belirli bir bölümlerine değerler atayabiliriz.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {15, 1, 99, 7, 7, -22, 11, 2, -49, 52}; Arrays.fill(liste, 2); System.out.println(Arrays.toString(liste)); int[] liste2 = {15, 1, 99, 7, 7, -22, 11, 2, -49, 52}; Arrays.fill(liste2, 3, 5, 7); System.out.println(Arrays.toString(liste2)); } } // Çıktısı // [2, 2, 2, 2, 2, 2, 2, 2, 2, 2] // [15, 1, 99, 7, 7, -22, 11, 2, -49, 52]

## Arrays.sort()

Arrays.sort() metodu ile dizilerdeki elemanları sıralayabiliriz.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {6, 1, 55, 21, 33, -321, -21, 2, -11, 27}; Arrays.sort(liste); System.out.println(Arrays.toString(liste)); } } // Çıktısı // [-321, -21, -11, 1, 2, 6, 21, 27, 33, 55]

## Arrays.binarySearch()

Java'da dizideki bir elemanın indis değerini bulmak için binarySearch kullanılabilir. Ama bu metodu kullanabilmek için, dizinin sıralı olması gerekmektedir.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {6, 1, 55, 21, 33, -321, -21, 2, -11, 27}; Arrays.sort(liste); System.out.println(Arrays.toString(liste)); int index = Arrays.binarySearch(liste, 33); System.out.println("33'ün indeksi :" + index); } } // [-321, -21, -11, 1, 2, 6, 21, 27, 33, 55] // 33'ün indeksi :8

## Arrays.copyOf() ve Arrays.copyOfRange() metotu

Mevcut diziden belli bir uzunlukta yeni bir dizi oluşturmak için Arrays.copyOf() metotu kullanılır

Mevcut diziden belli bir aralıkta yeni bir dizi oluşturmak için ise Array.copyOfRange() metodu kullanılır.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {6, 1, 55, 21, 33, -321, -21, 2, -11, 27}; int[] copyArray = Arrays.copyOf(liste, 3); System.out.println(Arrays.toString(copyArray)); int[] copyOfRangeArray = Arrays.copyOfRange(liste, 0,5); System.out.println(Arrays.toString(copyOfRangeArray)); } } // [6, 1, 55] // [6, 1, 55, 21, 33]

## Arrays.equals() metotu

Java'da iki dizinin eşitliğini kontrol etmek için Arrays.equals() metotu kullanılır.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] list1 = {1, 2, 3}; int[] list2 = {1, 2, 3}; int[] list3 = {1, 2, 10}; System.out.println(Arrays.equals(list1, list2)); // true System.out.println(Arrays.equals(list2, list3)); // false } }

# Arrays Sınıfı ve Metotları

Java.util paketindeki Arrays sınıfı, Java Collection Framework'ün bir parçasıdır. Bu sınıf, Java dizilerini dinamik olarak oluşturmak ve bunlara erişmek için statik metotlar sağlar. Yalnızca statik metotlar ve Object sınıfının metotlarından oluşur. Bu sınıfın metotları, sınıf adının kendisi tarafından kullanılabilir.

import java.util.Arrays;

Import deyimi ile java.util.Arrays sınıfını projeye dahil etmemiz gerekmektedir.

## Arrays.toString()

Diziye ait elemanları direk ekrana basmak için kullanılan bir metottur.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] dizi = {3, 5, 79, 12, 25, -3, 66, 82, -49, 152}; System.out.println(Arrays.toString(dizi)); } } // Çıktısı // [3, 5, 79, 12, 25, -3, 66, 82,-49,152]

## Arrays.fill()

Arrays.fill metodu ile dizilerimizin belirli bir bölümlerine değerler atayabiliriz.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {15, 1, 99, 7, 7, -22, 11, 2, -49, 52}; Arrays.fill(liste, 2); System.out.println(Arrays.toString(liste)); int[] liste2 = {15, 1, 99, 7, 7, -22, 11, 2, -49, 52}; Arrays.fill(liste2, 3, 5, 7); System.out.println(Arrays.toString(liste2)); } } // Çıktısı // [2, 2, 2, 2, 2, 2, 2, 2, 2, 2] // [15, 1, 99, 7, 7, -22, 11, 2, -49, 52]

## Arrays.sort()

Arrays.sort() metodu ile dizilerdeki elemanları sıralayabiliriz.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {6, 1, 55, 21, 33, -321, -21, 2, -11, 27}; Arrays.sort(liste); System.out.println(Arrays.toString(liste)); } } // Çıktısı // [-321, -21, -11, 1, 2, 6, 21, 27, 33, 55]

## Arrays.binarySearch()

Java'da dizideki bir elemanın indis değerini bulmak için binarySearch kullanılabilir. Ama bu metodu kullanabilmek için, dizinin sıralı olması gerekmektedir.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {6, 1, 55, 21, 33, -321, -21, 2, -11, 27}; Arrays.sort(liste); System.out.println(Arrays.toString(liste)); int index = Arrays.binarySearch(liste, 33); System.out.println("33'ün indeksi :" + index); } } // [-321, -21, -11, 1, 2, 6, 21, 27, 33, 55] // 33'ün indeksi :8

## Arrays.copyOf() ve Arrays.copyOfRange() metotu

Mevcut diziden belli bir uzunlukta yeni bir dizi oluşturmak için Arrays.copyOf() metotu kullanılır

Mevcut diziden belli bir aralıkta yeni bir dizi oluşturmak için ise Array.copyOfRange() metodu kullanılır.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] liste = {6, 1, 55, 21, 33, -321, -21, 2, -11, 27}; int[] copyArray = Arrays.copyOf(liste, 3); System.out.println(Arrays.toString(copyArray)); int[] copyOfRangeArray = Arrays.copyOfRange(liste, 0,5); System.out.println(Arrays.toString(copyOfRangeArray)); } } // [6, 1, 55] // [6, 1, 55, 21, 33]

## Arrays.equals() metotu

Java'da iki dizinin eşitliğini kontrol etmek için Arrays.equals() metotu kullanılır.

import java.util.Arrays; public class Main { public static void main(String[] args) { int[] list1 = {1, 2, 3}; int[] list2 = {1, 2, 3}; int[] list3 = {1, 2, 10}; System.out.println(Arrays.equals(list1, list2)); // true System.out.println(Arrays.equals(list2, list3)); // false } }

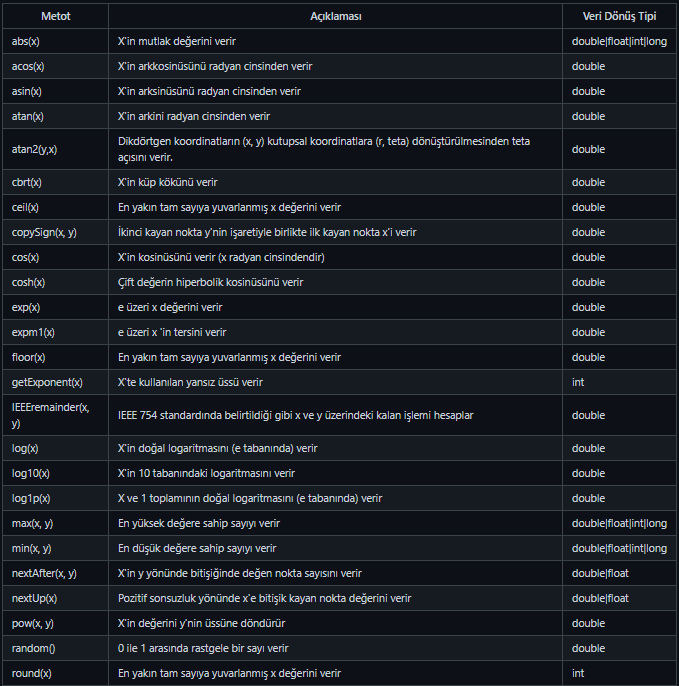
# Math Sınıfı ve Metotları

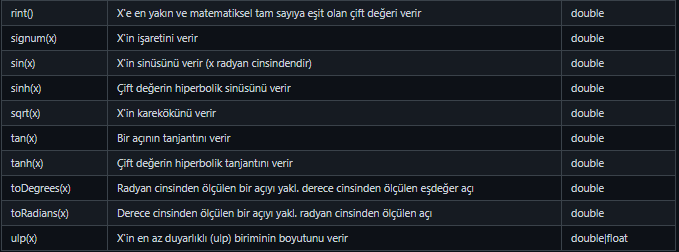
Java bazı durumları bizim için önceden yazdığı sınıflar ile bizlerin işini kolaylaştırmıştır. Bunlardan bir tanesi de Math sınıfıdır. Adında da anlaşılacağı gibi Java'da Math sınıfı matematiksel fonksiyonlar sunmaktadır. Java Math sınıfı, sayılar üzerinde matematiksel işlemler gerçekleştirmenize izin veren birçok metoda sahiptir.(

Math sınıfının kullanımı şu şekildedir;

Math.MethodAdi(Parametreler)

## Math Sınıfı Metotları



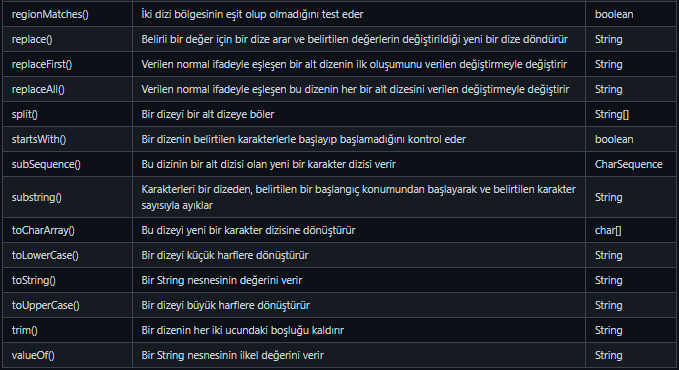


# String Sınıfı ve Metotları

Java'da String sınıfları **char**türünden verilerden oluşmuş bir kümedir yani karakterlerden oluşur. Java işlerimizi kolaylaştırmak ve String ifadelerle işlem yapmak için String sınıfını sunar.

## Java String Sınıfı Metotları





ve hatayla karşılaşılır. Buna da Syntax Error adı verilmektedir.