Java'ya Giriş ve İlk Program

Bilgisayar ve Programlamaya Giriş

Program | Araçlar | İlk program | Derleme | Hatalar

Sercan KÜLCÜ · Giresun Üniversitesi

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü:
https://sercankulcu.github.io/files/computer/slides/Bolum_05_Javaya_Giris_ve_Ilk_Program.html

GİRİŞ

Öğrenme Çıktıları

Program

Programı veri ve işlemler bütünü olarak açıklamak.

Java

Kaynak kod, bytecode ve JVM ilişkisini kurmak.

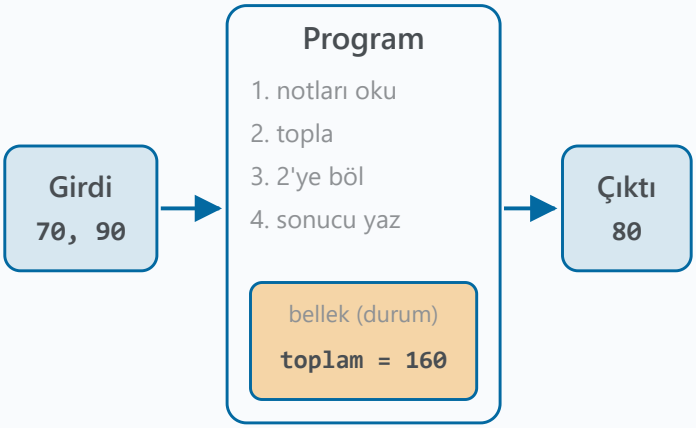
İlk program

Java dosyasını yazıp derlemek ve çalıştırmak.

Hatalar

Söz dizimi, çalışma zamanı ve mantık hatalarını ayırt etmek.

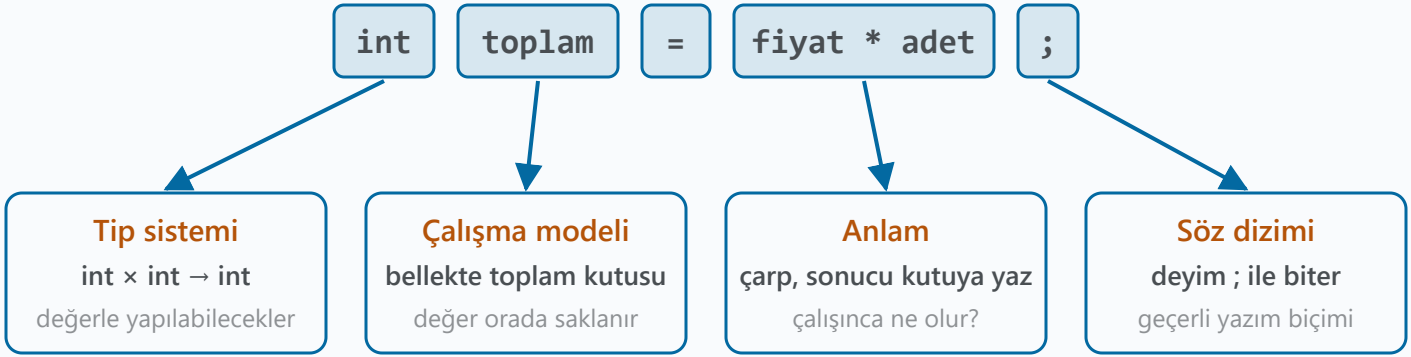
Program Nedir?



- Bilgisayarın belirli bir görevi yerine getirmesi için yürüttüğü komutlar bütünüdür.
- Girdi alabilir, veriyi işler, **durum** saklar (bellekteki değerler) ve çıktı üretir.
- Davranışı dilin kuralları ve yazılan algoritma belirler.
- Örnek: iki notun ortalaması → girdi 70, 90; çıktı 80.

TEMELLER

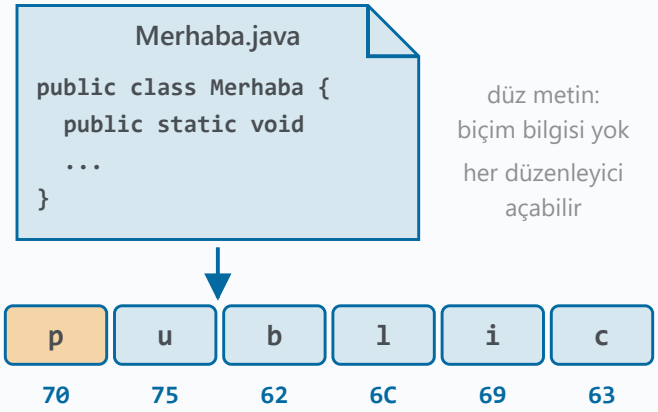
Programlama Dilinin Yönleri



- **Söz dizimi** geçerli biçimi, **anlam** çalıştırıldığında ne olacağını tanımlar.
- **Tip sistemi** işlemleri sınırlar; **çalışma modeli** belleğin ve komutların nasıl yürütüldüğünü belirler.
- Derleyici söz dizimini ve tipleri çalıştırmadan önce denetler.

KAYNAK KOD

Düz Metin Kaynak Dosyası



ASCII karakterler UTF-8'de 1 bayt (onaltılık); ç, ğ, ş gibi harfler 2 bayt

- .java dosyası düz metindir; Not Defteri, VS Code veya IDE ile açılabilir.
- Sürüm kontrol sistemleri satır değişikliklerini izleyebilir.
- Karakter kodlaması (UTF-8), satır sonu ve dosya adı tutarlı olmalıdır: Türkçe karakterler yanlış kodlamada bozulur.

KAYNAK KOD

Programlama Araçları

Düzenleyici
Kaynak kodu yazma.

IDE
Proje, tamamlama, derleme ve hata ayıklama.

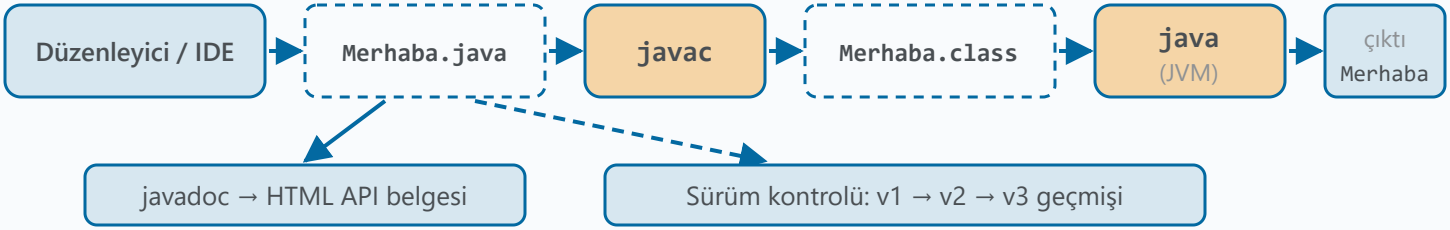
javac
Kaynak kodu bytecode'a derleme.

java
Sınıfı JVM üzerinde çalıştırma.

Javadoc
API belgesi üretme.

Sürüm kontrolü
Değişiklik geçmişi.

Araçlar bir Java programının yolculuğunda



JAVA

Java'nın Doğuşu ve Sürümleri



- James Gosling ve ekibi 1991'de Sun Microsystems'te başladı; dil 1995'te Java adıyla yayımlandı. 2010'dan beri Oracle geliştiriyor, açık kaynak kodu OpenJDK'dadır.
- Tasarım hedefi **bir kez yaz, her yerde çalıştır**: aynı bytecode her platformun JVM'sinde çalışır.
- Diğer özellikleri: nesne yönelimli, güçlü tip denetimi, otomatik bellek yönetimi (çöp toplama) ve geniş sınıf kütüphanesi.
- 2018'den beri her altı ayda bir yeni sürüm çıkar; **LTS** (uzun süreli destek) sürümleri yıllarca güncelleme aldığı için kurumlarda ve derslerde tercih edilir.

JAVA

Java Nerelerde Kullanılır?

Kurumsal sunucular

Bankacılık, e-ticaret ve kamu sistemlerinin arka ucu; Spring ve Jakarta EE çatıları.

Android

Uygulamalar Java ya da Kotlin ile yazılır; bytecode, telefondaki Android çalışma ortamına (ART) uygun biçime çevrilir.

Büyük veri

Hadoop ve Kafka gibi veri işleme araçları JVM üzerinde çalışır.

Geliştirme araçları

IntelliJ IDEA, Eclipse ve NetBeans gibi IDE'lerin kendisi de Java ile yazılmıştır.

Akıllı kartlar

SIM kartlar ve banka kartlarındaki küçük programlar Java Card ile çalışır.

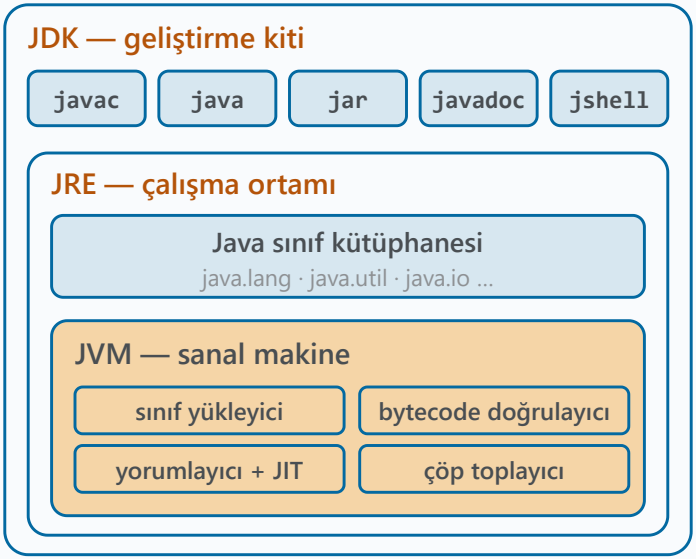
Oyun

Minecraft'ın Java Sürümü; masaüstü oyun ve simülasyonlar.

JVM yalnızca Java'yı değil, aynı bytecode'a derlenen Kotlin, Scala ve Groovy gibi dilleri de çalıştırır.

JAVA

JDK, JRE ve JVM



- **JVM** (Java Sanal Makinesi) bytecode'u yürüten sanal makinedir.
- **JRE** = JVM + sınıf kütüphanesi: programı çalıştırmak için gerekenler.
- **JDK** = JRE + geliştirme araçları (javac, jar, javadoc…). Geliştirici JDK kurar.
- Oracle, Java 11'den beri ayrı JRE sunmaz; programı yalnızca çalıştıracak kişi de JDK kurar (bazı dağıtıcılar JRE'yi ayrıca sunar).

JAVA

İlk Java Programı

```
public class Merhaba {

    public static void main(String[] args) {

        System.out.println("Merhaba");

    }

}
```

sınıf bildirimi
ad = dosya adı: Merhaba.java

main metodu
programın giriş noktası

deyim
konsola yaz; ; ile biter

} main bloğu biter

} sınıf bloğu biter

public
her yerden erişilebilir

static
nesne oluşturmadan çağrılır

void
değer döndürmez

String[] args
komut satırı argümanları

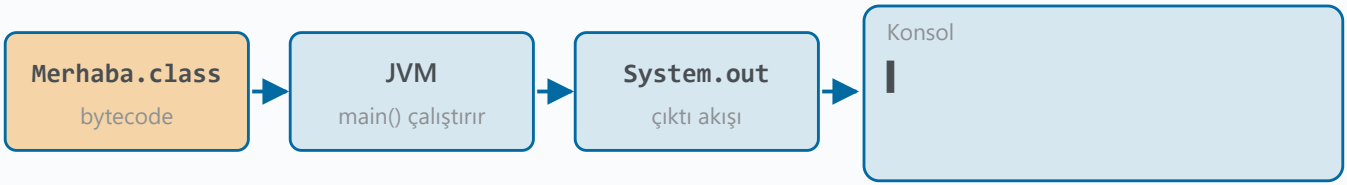
JAVA

`main` Metodunun İmzası

Yazılan (hepsi derlenir)	`java A` ile çalıştırınca
`public static void main(String[] args)`	**çalışır**
`public static void Main(String[] args)`	`Error: Main method not found in class A, …`
`public void main(String[] args)`	`Error: Main method is not static in class A, …`
`public static int main(String[] args)`	`Error: Main method must return a value of type void in class A, …`
`static void main(String[] args)`	`Error: Main method not found in class A, …`

- Derleyici için hepsi sıradan bir metottur; bu yüzden hata **derlemede değil, başlatırken** çıkar. JVM giriş noktasını adı, parametresi ve dönüş tipiyle arar.
- Parametre adı serbesttir (`String[] a`); `String args[]` yazımı da geçerlidir. `main` küçük harfle yazılır: Java büyük-küçük harfe duyarlıdır.
- İletiler JDK 21 ile alınmıştır. Java 25'ten itibaren public ya da static olmayan `main` da başlatılabilir; bu derste klasik imza kullanılır.

Kod İzleme: `Merhaba`



`java Merhaba` komutuyla JVM başlar ve **Merhaba** sınıfını (Merhaba.class) belleğe yükler.

Adım 1 / 6

```
▶ public class Merhaba {
2     public static void main(String[] args) {
3         System.out.println("Merhaba");
4     }
5 }
```

DEĞİŞKENLER

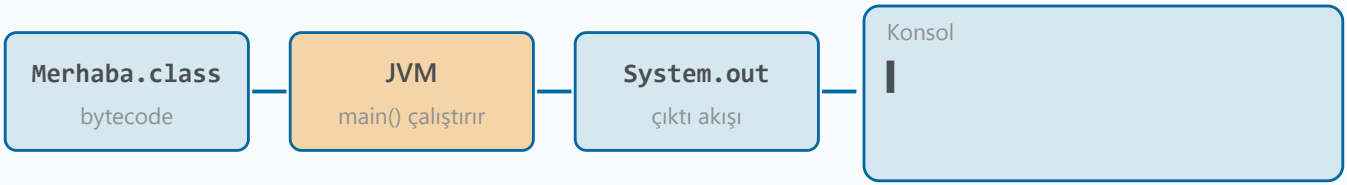
—

ÇAĞRI YIĞINI

boş

ÇIKTI

Kod İzleme: `Merhaba`



JVM **main** metodunu çağırır: yığında main çerçevesi açılır. Komut satırında argüman verilmediği için args boştur.

Adım 2 / 6

```
1 public class Merhaba {
▶     public static void main(String[] args) {
3         System.out.println("Merhaba");
4     }
5 }
```

DEĞİŞKENLER

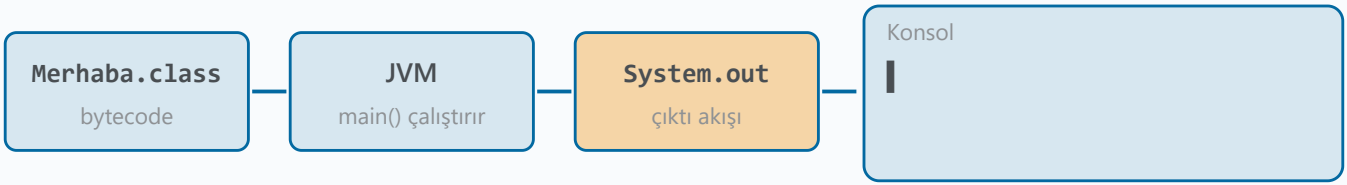
`args = boş (argüman yok)`

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

Kod İzleme: `Merhaba`



Deyim çalışır: `"Merhaba"` metni `System.out.println` metoduna gönderilir.

Adım 3 / 6

```
1 public class Merhaba {
2     public static void main(String[] args) {
▶         System.out.println("Merhaba");
4     }
5 }
```

DEĞİŞKENLER

`args = boş (argüman yok)`

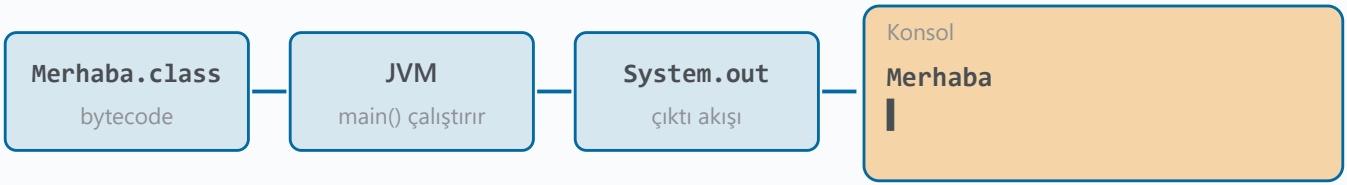
ÇAĞRI YIĞINI

`println("Merhaba")`

`main(args)`

ÇIKTI

Kod İzleme: `Merhaba`



println metni yazar ve **satır sonu** ekler: imleç alt satıra geçer. Metot biter, main'e dönülür.

Adım 4 / 6

```
1 public class Merhaba {
2     public static void main(String[] args) {
▶         System.out.println("Merhaba");
4     }
5 }
```

DEĞİŞKENLER

`args = boş (argüman yok)`

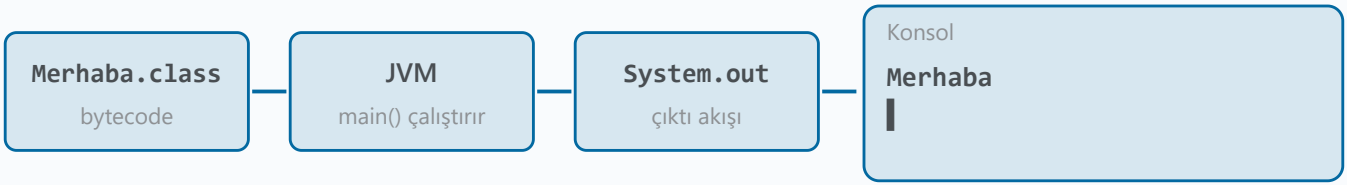
ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

```
Merhaba
```

Kod İzleme: `Merhaba`



main bloğunun `}` karakterine gelindi: main biter, çerçevesi yığından kaldırılır.

Adım 5 / 6

```
1 public class Merhaba {
2     public static void main(String[] args) {
3         System.out.println("Merhaba");
▶     }
5 }
```

DEĞİŞKENLER

`args = boş (argüman yok)`

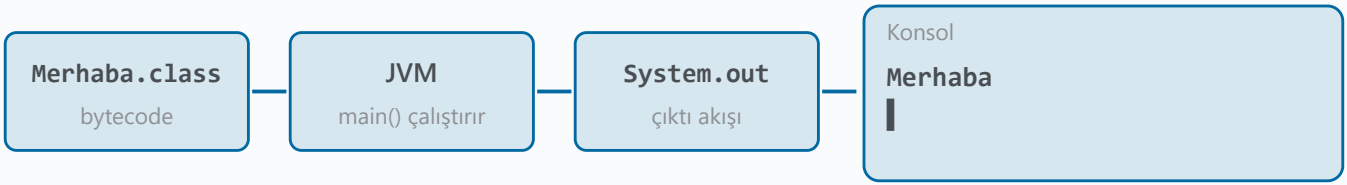
ÇAĞRI YIĞINI

boş

ÇIKTI

```
Merhaba
```

Kod İzleme: `Merhaba`



Çalışacak başka kod yok: JVM kapanır, program **sonlanır**. Toplam 1 deyim çalıştı, 1 satır çıktı üretildi.

Adım 6 / 6

```
1 public class Merhaba {
2     public static void main(String[] args) {
3         System.out.println("Merhaba");
4     }
▶ }
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

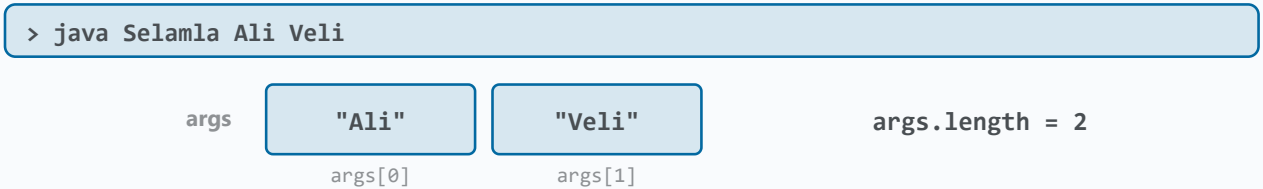
boş

ÇIKTI

```
Merhaba
```

JAVA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Komut Satırı Argümanları



Sınıf adından sonra yazılan sözcükler **komut satırı argümanlarıdır**. JVM bunları bir String dizisine koyup **main**'e verir.

Adım 1 / 8

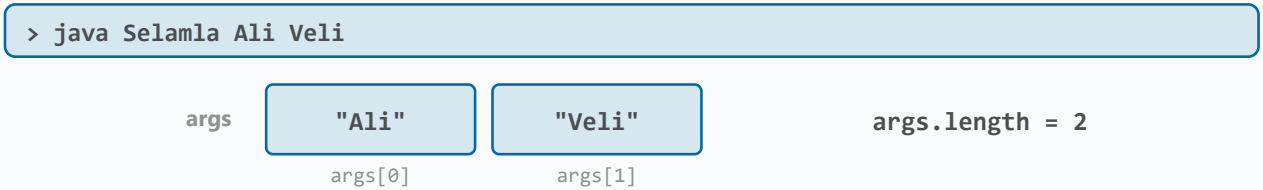
```
1 public class Selamla {
▶     public static void main(String[] args) {
3         System.out.println(args.length);
4         System.out.println("Selam " + args[0]);
5         System.out.println("Selam " + args[1]);
6     }
7 }
```

DEĞİŞKENLER

`args = ["Ali", "Veli"]`

ÇIKTI

Kod İzleme: Komut Satırı Argümanları



args.length dizideki argüman sayısıdır: 2.

Adım 2 / 8

```
1 public class Selamla {
2     public static void main(String[] args) {
▶           System.out.println(args.length);
4           System.out.println("Selam " + args[0]);
5           System.out.println("Selam " + args[1]);
6     }
7 }
```

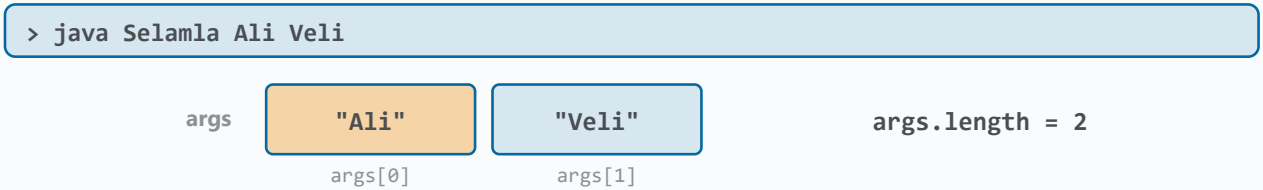
DEĞİŞKENLER

`args = ["Ali", "Veli"]`

ÇIKTI

```
2
```

Kod İzleme: Komut Satırı Argümanları



İlk argüman **args[0]**'dadır (indis 0'dan başlar): Selam Ali.


```
1 public class Selamla {
2     public static void main(String[] args) {
3         System.out.println(args.length);
▶         System.out.println("Selam " + args[0]);
5         System.out.println("Selam " + args[1]);
6     }
7 }
```

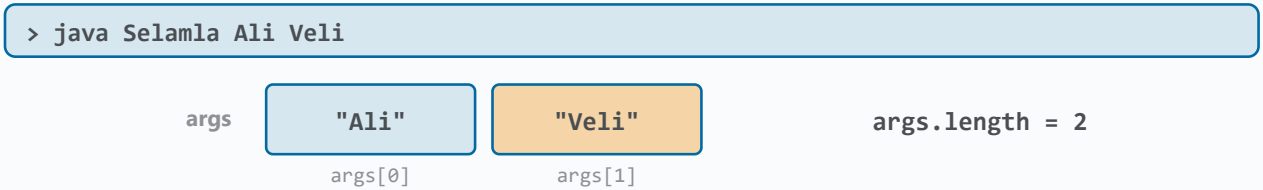
DEĞİŞKENLER

`args = ["Ali", "Veli"]`

ÇIKTI

```
2
Selam Ali
```

Kod İzleme: Komut Satırı Argümanları



args[1] → Veli. main biter, program sonlanır.

Adım 4 / 8

```
1 public class Selamla {
2     public static void main(String[] args) {
3         System.out.println(args.length);
4         System.out.println("Selam " + args[0]);
▶         System.out.println("Selam " + args[1]);
6     }
7 }
```

DEĞİŞKENLER

`args = ["Ali", "Veli"]`

ÇIKTI

```
2
Selam Ali
Selam Veli
```

Kod İzleme: Komut Satırı Argümanları

```
> java Selamla Ali
```



İkinci çalıştırma: aynı .class, bu kez tek argüman. Kod değişmedi, yeniden derlemek gerekmez.

Adım 5 / 8

```
1 public class Selamla {
▶     public static void main(String[] args) {
3         System.out.println(args.length);
4         System.out.println("Selam " + args[0]);
5         System.out.println("Selam " + args[1]);
6     }
7 }
```

DEĞİŞKENLER

`args = ["Ali"]`

ÇIKTI

Kod İzleme: Komut Satırı Argümanları

```
> java Selamla Ali
```



args.length = 1.

Adım 6 / 8

```
1 public class Selamla {
2     public static void main(String[] args) {
▶         System.out.println(args.length);
4         System.out.println("Selam " + args[0]);
5         System.out.println("Selam " + args[1]);
6     }
7 }
```

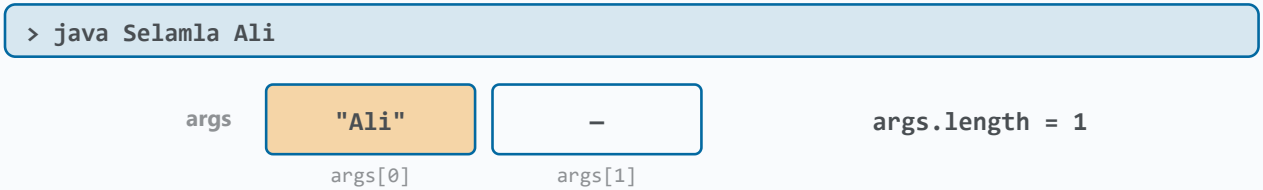
DEĞİŞKENLER

`args = ["Ali"]`

ÇIKTI

```
1
```

Kod İzleme: Komut Satırı Argümanları



args[0] var: Selam Ali.

Adım 7 / 8

```
1 public class Selamla {
2     public static void main(String[] args) {
3         System.out.println(args.length);
▶        System.out.println("Selam " + args[0]);
5         System.out.println("Selam " + args[1]);
6     }
7 }
```

DEĞİŞKENLER

`args = ["Ali"]`

ÇIKTI

```
1
Selam Ali
```

Kod İzleme: Komut Satırı Argümanları



args[1] yok: JVM **ArrayIndexOutOfBoundsException** fırlatır ve program durur. Derleyici bunu bilemezdi; hata çalışırken, girdiye bağlı olarak çıktı.

Adım 8 / 8

```
1 public class Selamla {
2     public static void main(String[] args) {
3         System.out.println(args.length);
4         System.out.println("Selam " + args[0]);
▶         System.out.println("Selam " + args[1]);
6     }
7 }
```

DEĞİŞKENLER

`args = ["Ali"]`

ÇIKTI

```
1
Selam Ali
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
: Index 1 out of bounds for length 1
    at Selamla.main(Selamla.java:5)
```

Kod İzleme: `print` ve `println`

Konsol

imleç: ▌
satır 1, sütun 1

JVM **main** metodunu çağırır; konsol henüz boş, imleç 1. satırın başında.

Adım 1 / 5

```
1 public class Selam {
▶     public static void main(String[] args) {
3         System.out.print("Merhaba ");
4         System.out.println("Dünya");
5         System.out.println("Java");
6     }
7 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

Kod İzleme: `print` ve `println`

Konsol

```
Merhaba
```

imleç: ▌
satır 1, sütun 9

print "Merhaba " yazar ama **satır sonu eklemez**: imleç aynı satırda, boşluğun hemen arkasında kalır.

Adım 2 / 5

```
1 public class Selam {
2     public static void main(String[] args) {
▶         System.out.print("Merhaba ");
4         System.out.println("Dünya");
5         System.out.println("Java");
6     }
7 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

```
Merhaba
```

Kod İzleme: `print` ve `println`

Konsol

```
Merhaba Dünya
▌
```

imleç: ▌
satır 2, sütun 1

println "Dünya" yazar ve **satır sonu** ekler: sonraki çıktı alt satırdan başlar.

Adım 3 / 5

```
1 public class Selam {
2     public static void main(String[] args) {
3         System.out.print("Merhaba ");
▶         System.out.println("Dünya");
5         System.out.println("Java");
6     }
7 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

```
Merhaba Dünya
```

Kod İzleme: `print` ve `println`

Konsol

```
Merhaba Dünya
Java
```

imleç: ▌
satır 3, sütun 1

println "Java" yazar ve **satır sonu** ekler: sonraki çıktı alt satırdan başlar.

Adım 4 / 5

```
1 public class Selam {
2     public static void main(String[] args) {
3         System.out.print("Merhaba ");
4         System.out.println("Dünya");
▶         System.out.println("Java");
6     }
7 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

```
Merhaba Dünya
Java
```

Kod İzleme: `print` ve `println`

Konsol

```
Merhaba Dünya
Java
```

imleç: ▌
satır 3, sütun 1

main biter; konsolda **2 satır** var: 3 çağrı yapıldı ama yalnızca println satırı bitirdi.

Adım 5 / 5

```
1 public class Selam {
2     public static void main(String[] args) {
3         System.out.print("Merhaba ");
4         System.out.println("Dünya");
5         System.out.println("Java");
▶       }
7 }
```

ÇAĞRI YIĞINI

boş

ÇIKTI

```
Merhaba Dünya
Java
```

JAVA

Kaçış Dizileri

Dizi	Anlamı	Örnek	Çıktı
`\n`	yeni satır	`"A\nB"`	A, alt satırda B
`\t`	sekme (tab)	`"Ad\tNot"`	Ad Not
`\"`	çift tırnak	`"\"Java\""`	"Java"
`\\`	ters eğik çizgi	`"C:\\ders"`	C:\ders
`\'`	tek tırnak (char içinde)	`'\''`	'

- Dizgi içinde yazılamayan veya özel anlamı olan karakterler `\` ile başlayan dizilerle yazılır.
- Tırnak içindeki `"` metni bitirir; metnin parçası olacaksa `\"` yazılır.

Kod İzleme: Kaçış Dizileri

Konsol

main başlar; konsol boş.

Adım 1 / 5

```
1 public class Kacis {
▶     public static void main(String[] args) {
3         System.out.println("Ad\tNot");
4         System.out.println("Ali\t85");
5         System.out.println("\"Java\" dili");
6         System.out.print("Satır 1\nSatır 2\n");
7     }
8 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

Kod İzleme: Kaçış Dizileri

Konsol

```
Ad      Not
```

`\t` imleci bir sonraki **sekme durağına** (9. sütun) taşır.

Adım 2 / 5

```
1 public class Kacis {
2     public static void main(String[] args) {
▶           System.out.println("Ad\tNot");
4           System.out.println("Ali\t85");
5           System.out.println("\"Java\" dili");
6           System.out.print("Satır 1\nSatır 2\n");
7     }
8 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

```
Ad      Not
```

Kod İzleme: Kaçış Dizileri

Konsol

```
Ad      Not
Ali     85
```

Sekme sayesinde 85, Not başlığının altına hizalanır: basit tablo.

Adım 3 / 5

```
1 public class Kacis {
2     public static void main(String[] args) {
3         System.out.println("Ad\tNot");
▶         System.out.println("Ali\t85");
5         System.out.println("\"Java\" dili");
6         System.out.print("Satır 1\nSatır 2\n");
7     }
8 }
```

ÇAĞRI YIĞINI

```
main(args)
```

ÇIKTI

```
Ad      Not
Ali     85
```

Kod İzleme: Kaçış Dizileri

Konsol

```
Ad      Not
Ali     85
"Java" dili
```

`\"` metni bitirmeden çift tırnak yazar.

Adım 4 / 5

```
1 public class Kacis {
2     public static void main(String[] args) {
3         System.out.println("Ad\tNot");
4         System.out.println("Ali\t85");
▶         System.out.println("\"Java\" dili");
6         System.out.print("Satır 1\nSatır 2\n");
7     }
8 }
```

ÇAĞRI YIĞINI

`main(args)`

ÇIKTI

```
Ad      Not
Ali     85
"Java" dili
```

Kod İzleme: Kaçış Dizileri

Konsol

```
"Java" dili
Satır 1
Satır 2
```

Tek print iki satır üretti: `\n` satır sonu karakteridir; println de satır sonu ekler (Windows'ta `\r\n`).

Adım 5 / 5

```
1 public class Kacis {
2     public static void main(String[] args) {
3         System.out.println("Ad\tNot");
4         System.out.println("Ali\t85");
5         System.out.println("\"Java\" dili");
▶         System.out.print("Satır 1\nSatır 2\n");
7     }
8 }
```

ÇAĞRI YIĞINI

```
main(args)
```

ÇIKTI

```
Ad      Not
Ali     85
"Java" dili
Satır 1
Satır 2
```

JAVA

Kaçış Dizisi Tuzakları



Deyim	Sonuç
`System.out.println("C:\yeni");`	derleme hatası: `illegal escape character` ; `\y` diye bir kaçış dizisi yok
`System.out.println("C:\new");`	derlenir ama iki satır yazar: `C:` ve `ew`
`System.out.println("C:\\yeni");`	`C:\yeni` : ters eğik çizgi iki kez yazılır
`System.out.println("Ali'nin notu");`	`Ali'nin notu` : dizgi içinde tek tırnak kaçışsız yazılır
`System.out.println("\u00C7ay");`	`Çay` : `\u` ve dört onaltılık basamak bir Unicode karakteri verir

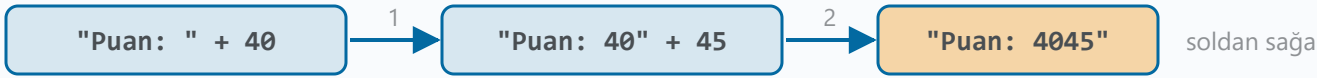
Windows dosya yolu yazarken en sık yapılan hata tek ters eğik çizgidir; Java dosya işlemlerinde `"C:/yeni"` biçimi de kabul edilir.

JAVA

Dizgi Birleştirme: `+` İki İş Yapar

- İki sayı arasında `+` **toplar**; işlenenlerden biri String ise **birleştirir**: sayı metne çevrilip sona eklenir.
- Aynı öncelikteki er soldan sağa değerlendirilir; `*` ve `/` her zaman `+`'dan önce yapılır.

```
System.out.println("Puan: " + 40 + 45);
```



İfade	Çıktı	Neden?
`40 + 45 + " puan"`	`85 puan`	önce iki int toplanır, sonra metin eklenir
`"Puan: " + 40 + 45`	`Puan: 4045`	en soldaki String; ikisi de metne eklenir
`"Puan: " + (40 + 45)`	`Puan: 85`	parantez önce hesaplanır
`"Ort: " + 85 / 2`	`Ort: 42`	`/` önce; iki int bölününce kesir atılır

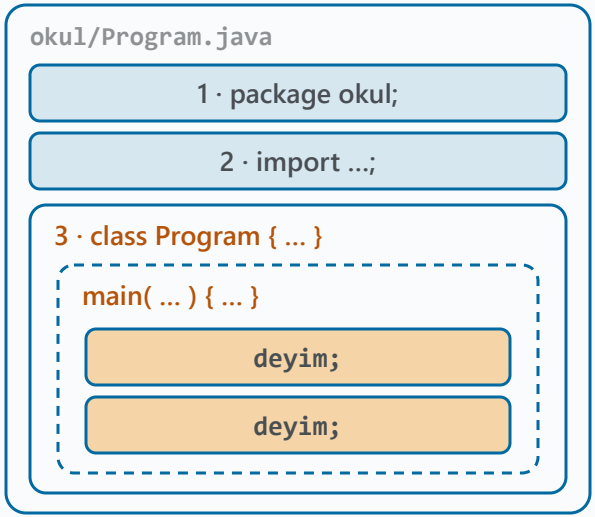
KAYNAK KOD

Java Dosyasının Yapısı

```java
package okul;

import java.util.Scanner;

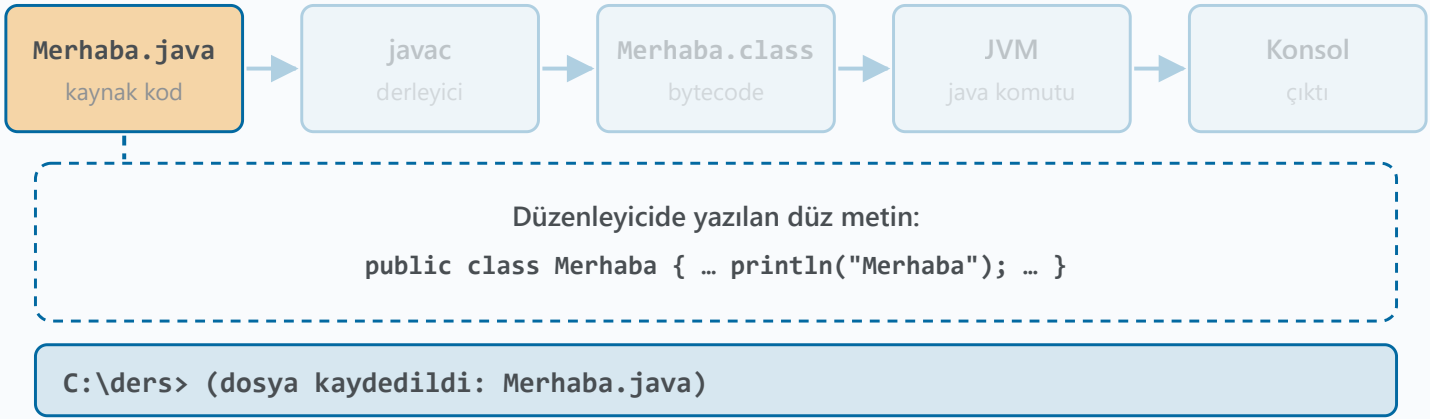
public class Program {
    public static void main(String[] args) {
        System.out.println("Merhaba");
    }
}
```



- package ve import bildirimleri sınıftan önce gelir; public sınıf adı dosya adıyla eşleşir.
- `package okul;` → dosya `okul/` klasöründe durur; çalıştırma: `java okul.Program`.

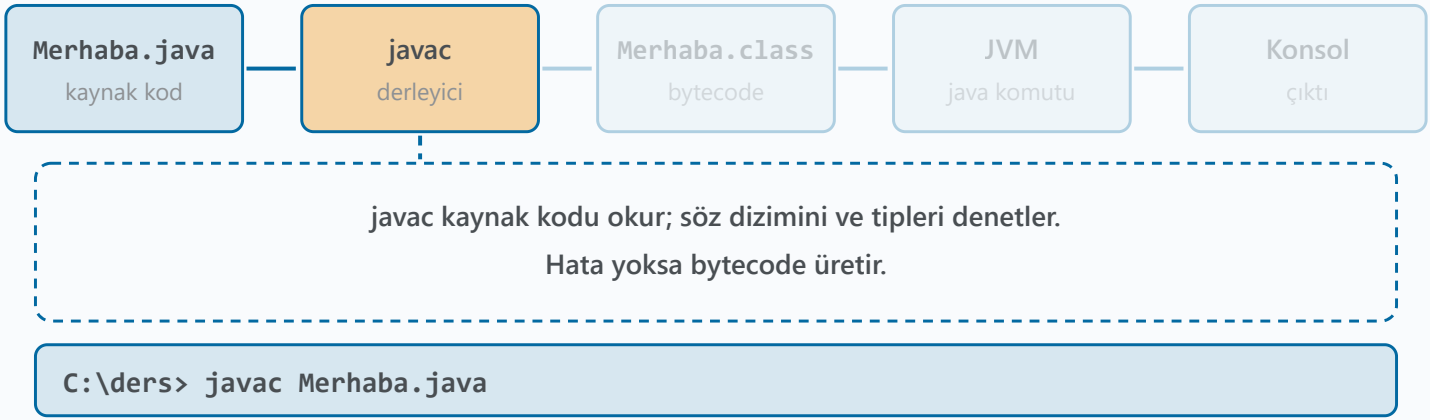
ÇALIŞTIRMA

Derleme Hattı Adım Adım



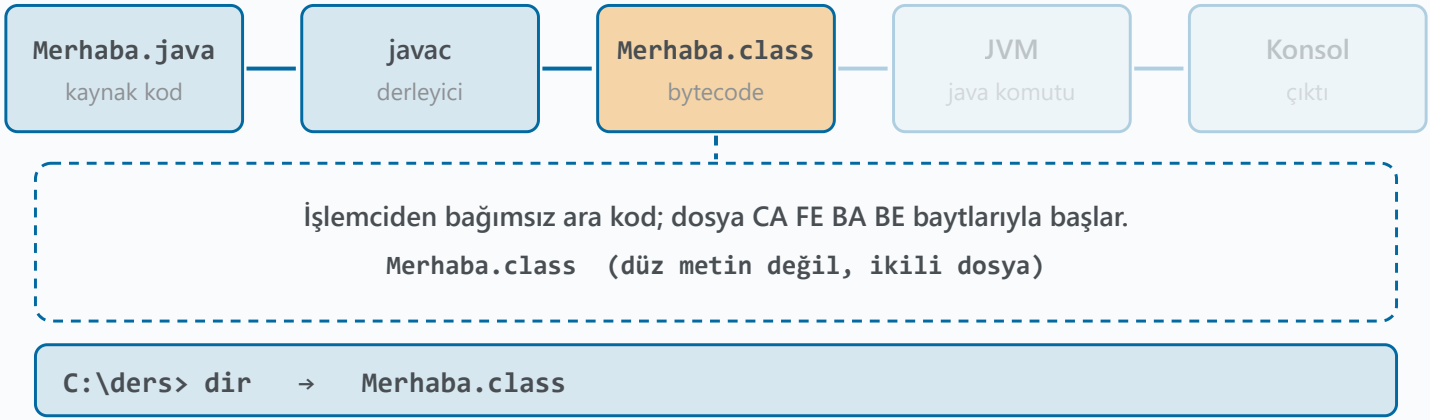
1. Yaz: kaynak kod `Merhaba.java` adıyla kaydedilir. Dosya adı public sınıf adıyla aynı olmalıdır.

Derleme Hattı Adım Adım



2. Derle: `javac` (JDK'daki derleyici) tüm dosyayı çalıştırmadan önce denetler.

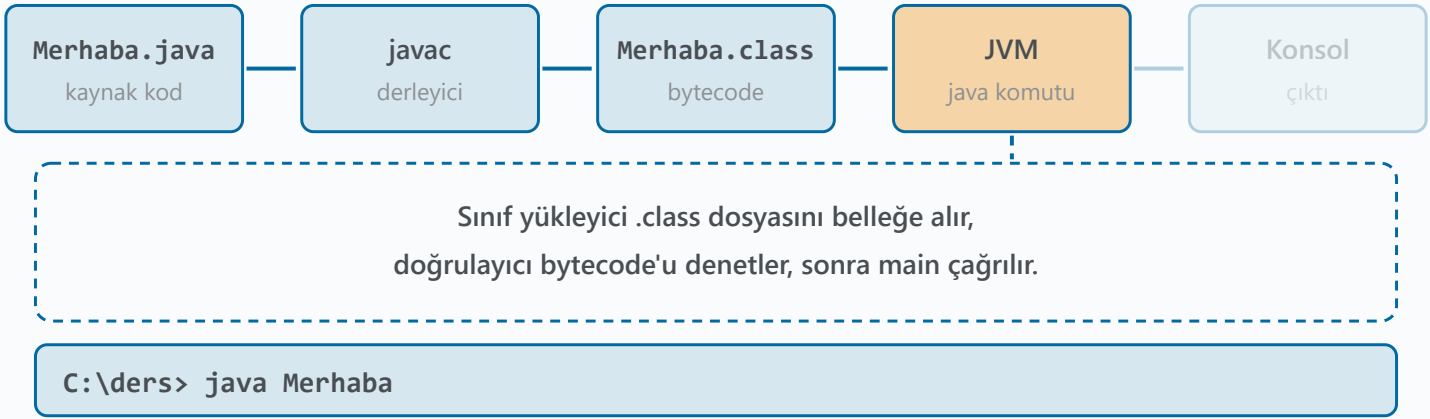
Derleme Hattı Adım Adım



3. Bytecode: derleme başarılıysa aynı klasörde `Merhaba.class` oluşur. Bu dosya Windows, Linux ve macOS'ta aynıdır.

ÇALIŞTIRMA

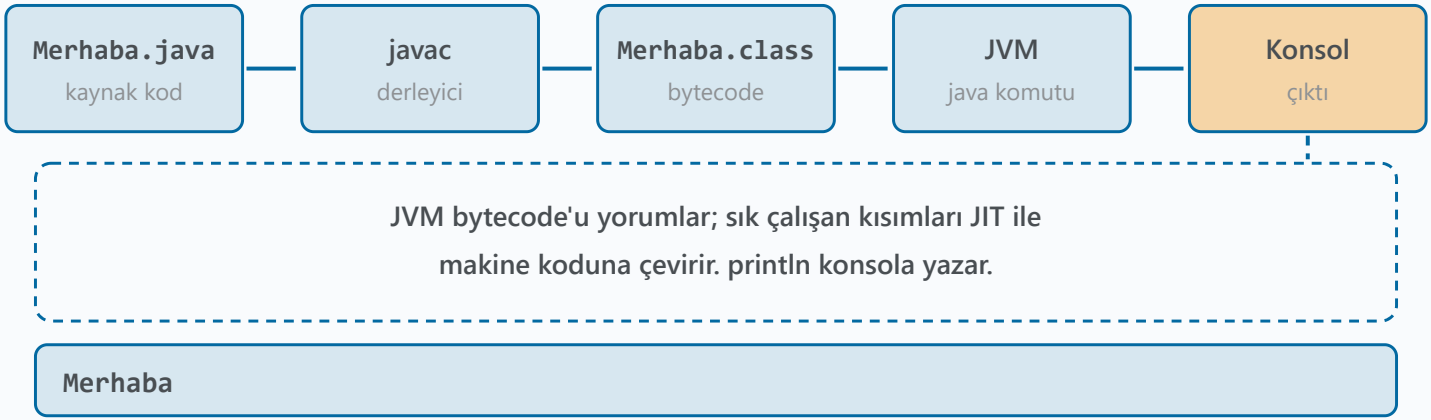
Derleme Hattı Adım Adım



4. Çalıştır: `java Merhaba` JVM'yi başlatır. Sınıf adı yazılır; **.class uzantısı yazılmaz**.

ÇALIŞTIRMA

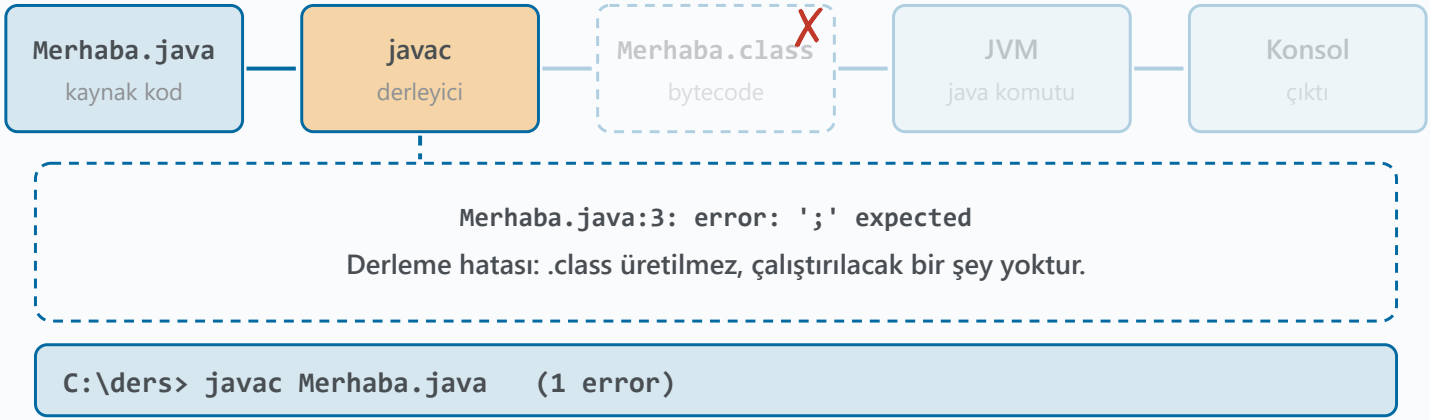
Derleme Hattı Adım Adım



5. Çıktı: konsolda `Merhaba` görünür. Değişiklik yaparsanız yeniden **derlemeniz** gerekir.

ÇALIŞTIRMA

Derleme Hattı Adım Adım



Hata olursa: javac satır numarasıyla hata bildirir ve **.class üretmez**. Önce derleme hataları giderilir.

ÇALIŞTIRMA

JDK Kurulumu ve Komutlar

Komut	Ne yapar?
`java -version`	kurulu Java sürümünü gösterir
`javac -version`	derleyici sürümünü gösterir (**javac** yalnız JDK içindedir)
`javac Merhaba.java`	derler: `Merhaba.class` üretir
`java Merhaba`	çalıştırır (uzantı yazılmaz)
`java Merhaba.java`	Java 11+: derle ve çalıştır tek adımda, .class bırakmaz

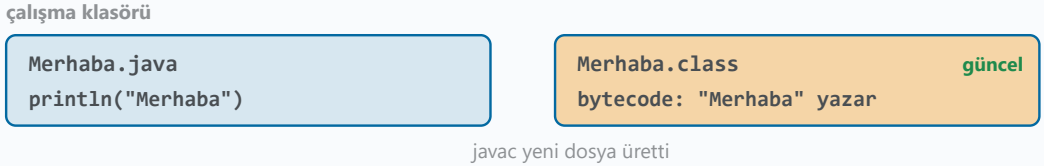
PATH ve JAVA_HOME

Komutlar bulunamıyorsa JDK'nın `bin` klasörünü PATH ortam değişkenine ekle. `JAVA_HOME` JDK klasörünü gösterir; araçlar bunu kullanır.

Windows konsolunda Türkçe

Derlerken `javac -encoding UTF-8 Merhaba.java`. Çıktıda harfler bozuksa konsolda önce `chcp 65001` çalıştır.

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



Derleme başarılı olduğunda javac **hiçbir şey yazmaz**; sessizce Merhaba.class oluşur.

Adım 1 / 7

```
▶ > javac Merhaba.java
2 > java Merhaba
3   (düzenleyicide "Merhaba" → "Selam", kaydet)
4 > java Merhaba
5 > javac Merhaba.java
6 > java Merhaba
7 > java Merhaba.class
```

DEĞİŞKENLER

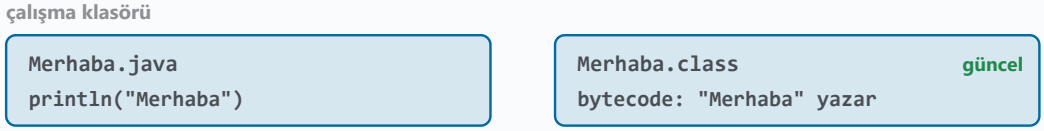
`Merhaba.java = Merhaba`

`Merhaba.class = Merhaba`

ÇIKTI

```
> javac Merhaba.java
```

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



JVM Merhaba.class'ı yükler ve çalıştırır.

Adım 2 / 7

```
1 > javac Merhaba.java
▶ > java Merhaba
3   (düzenleyicide "Merhaba" → "Selam", kaydet)
4 > java Merhaba
5 > javac Merhaba.java
6 > java Merhaba
7 > java Merhaba.class
```

DEĞİŞKENLER

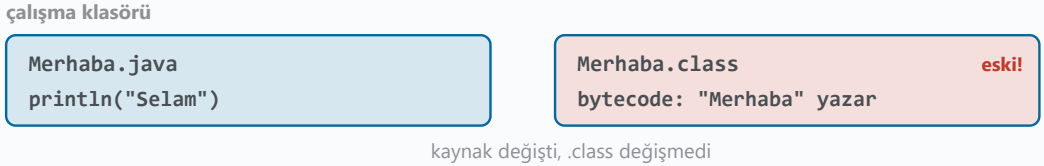
`Merhaba.java = Merhaba`

`Merhaba.class = Merhaba`

ÇIKTI

```
> java Merhaba
Merhaba
```

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



Kaynak düzenlendi ve kaydedildi. Ama .class dosyası **kendiliğinden** güncellenmez.


```
1 > javac Merhaba.java
2 > java Merhaba
  (düzenleyicide "Merhaba" → "Selam", kaydet)
4 > java Merhaba
5 > javac Merhaba.java
6 > java Merhaba
7 > java Merhaba.class
```

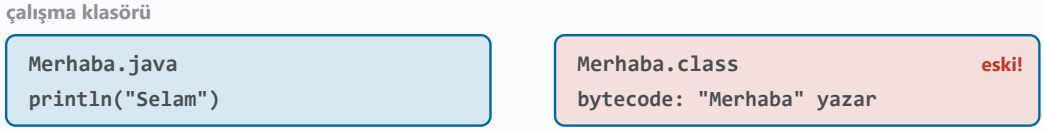
DEĞİŞKENLER

`Merhaba.java = Selam`

`Merhaba.class = Merhaba (eski)`

ÇIKTI

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



Çıktı hâlâ **Merhaba**: java komutu kaynağa bakmaz, yalnızca .class'taki bytecode'u çalıştırır.


```
1 > javac Merhaba.java
2 > java Merhaba
3   (düzenleyicide "Merhaba" → "Selam", kaydet)
▶ > java Merhaba
5 > javac Merhaba.java
6 > java Merhaba
7 > java Merhaba.class
```

DEĞİŞKENLER

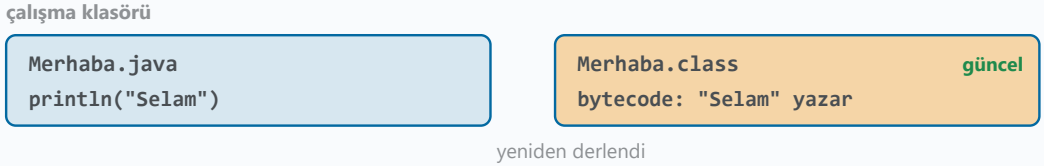
`Merhaba.java` = **Selam**

`Merhaba.class` = **Merhaba (eski)**

ÇIKTI

```
> java Merhaba
Merhaba
```

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



Yeniden derleme .class dosyasını yeniler.


```
1 > javac Merhaba.java
2 > java Merhaba
3   (düzenleyicide "Merhaba" → "Selam", kaydet)
4 > java Merhaba
▶ > javac Merhaba.java
6 > java Merhaba
7 > java Merhaba.class
```

DEĞİŞKENLER

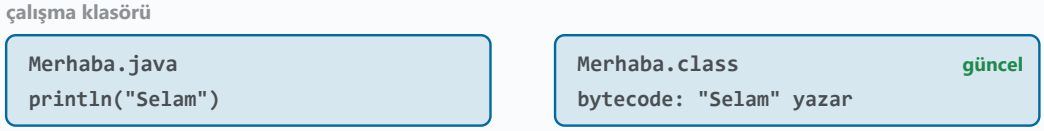
`Merhaba.java = Selam`

`Merhaba.class = Selam`

ÇIKTI

```
> javac Merhaba.java
```

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



Değişiklik artık görünüyor: **Selam**.

Adım 6 / 7

```
1 > javac Merhaba.java
2 > java Merhaba
3   (düzenleyicide "Merhaba" → "Selam", kaydet)
4 > java Merhaba
5 > javac Merhaba.java
▶ > java Merhaba
7 > java Merhaba.class
```

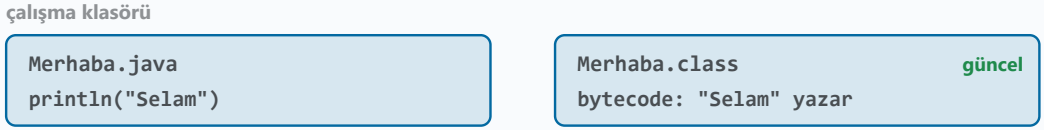
DEĞİŞKENLER

- `Merhaba.java = Selam`
- `Merhaba.class = Selam`

ÇIKTI

```
> java Merhaba
Selam
```

Konsol Oturumu: Derle, Değiştir, Yeniden Derle



java bir sınıf adı bekler, dosya adı değil

java'ya **dosya adı** verildi: JVM "Merhaba.class" adlı bir sınıf arar, bulamaz. Uzantı yazılmaz.

Adım 7 / 7

```
1 > javac Merhaba.java
2 > java Merhaba
3   (düzenleyicide "Merhaba" → "Selam", kaydet)
4 > java Merhaba
5 > javac Merhaba.java
6 > java Merhaba
▶ > java Merhaba.class
```

DEĞİŞKENLER

`Merhaba.java = Selam`

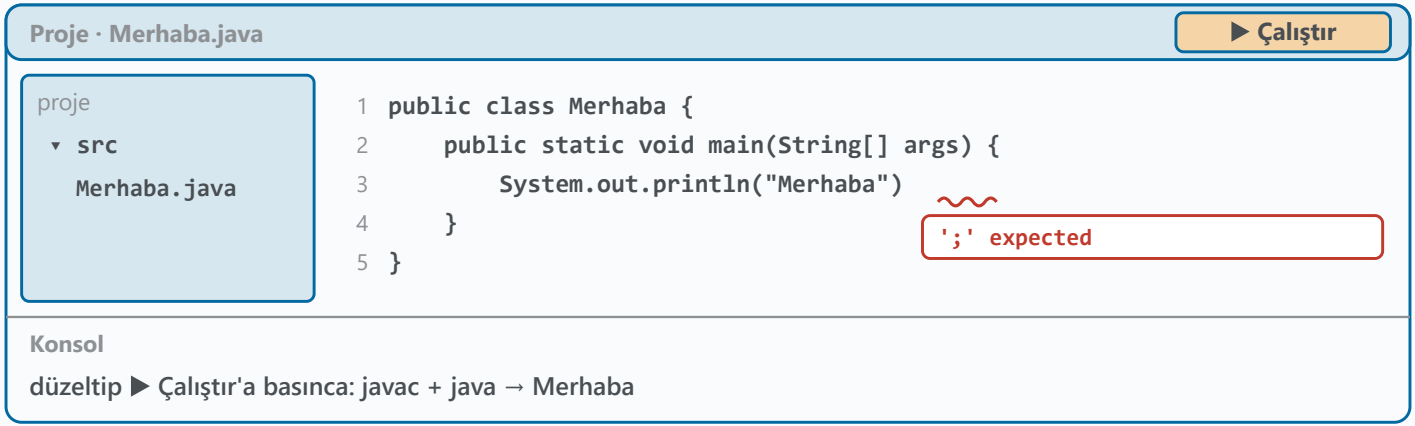
`Merhaba.class = Selam`

ÇIKTI

```
> java Merhaba.class
Error: Could not find or load main class
Merhaba.class
Caused by:
java.lang.ClassNotFoundException:
Merhaba.class
```

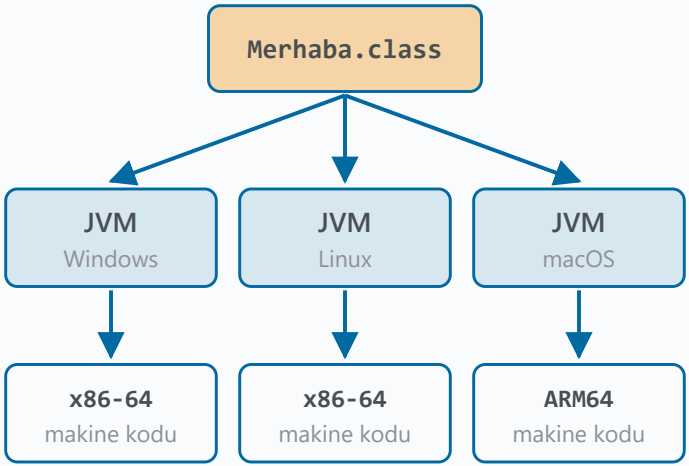
ÇALIŞTIRMA

IDE ile Çalışmak



- ▶ **Çalıştır** düğmesi arka planda derleme ve çalıştırma adımlarını yapar; değiştirilen dosya önce yeniden derlenir, eski .class sorunu yaşanmaz.
- Derleme hataları daha yazarken kırmızı alt çizgiyle işaretlenir; imleç üzerine gelince derleyicinin iletisi görünür.
- Kod tamamlama, otomatik girinti, yeniden adlandırma ve kesme noktalı hata ayıklayıcı sunar. Yaygın Java IDE'leri: IntelliJ IDEA, Eclipse, NetBeans; VS Code Java eklentileriyle.
- IDE kolaylık sağlar, ama arka planda olanı bilmek IDE'nin olmadığı ortamlarda (sunucu, uzak bağlantı) hata iletisini anlamayı sağlar.

Bytecode ve JVM



bir kez derle, JVM olan her yerde çalıştır

- Derleyici işlemciye özel makine kodu yerine **bytecode** üretir.
- Her platformun kendi JVM'si bytecode'u yorumlar; sık çalışan kısımları JIT ile o işlemcinin makine koduna çevirir.
- Sınıf yükleme, doğrulama ve bellek yönetimi (çöp toplama) çalışma ortamının parçalarıdır.

ÇALIŞTIRMA

Bytecode'un İçi: `javap`

```
> javap -c Merhaba
...
public static void main(java.lang.String[]);
  Code:
     0: getstatic     // System.out
     3: ldc           // "Merhaba"
     5: invokevirtual // println
     8: return
```

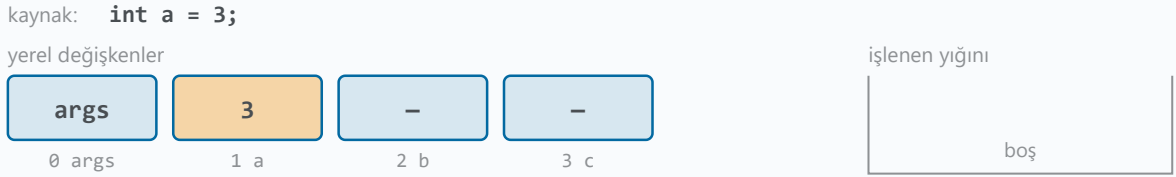
javap sabit havuzu göstergelerini de yazar; burada kısaltıldı.

- `javap -c` JDK ile gelir; .class dosyasındaki komutları okunur biçimde listeler.
- **getstatic** System.out nesnesini, **ldc** "Merhaba" sabitini **işlenen yığınına** koyar.
- **invokevirtual** ikisini yığından alıp println'i çağırır; **return** main'i bitirir.
- Her komutun işlem kodu bir bayttır; "bytecode" adı buradan gelir. Soldaki sayılar komutun bayt konumudur.

JVM bir **yığın makinesidir**: işlemciler gibi yazmaç kullanmak yerine değerleri bir işlenen yığınına koyar, işlemleri yığının tepesindeki değerlerle yapar.

ÇALIŞTIRMA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: JVM Bytecode'u Yürütüyor



Kaynaktaki dört satır javap -c ile görülen 12 komuta derlendi. **iconst_3** yığına 3 koyar, **istore_1** onu alıp 1 numaralı yerel değişkene (a) yazar.

Adım 1 / 10

```
▶  0: iconst_3
▶  1: istore_1
   2: iconst_4
   3: istore_2
   4: iload_1
   5: iload_2
   6: iadd
   7: istore_3
   8: getstatic     // System.out
  11: iload_3
  12: invokevirtual // println(int)
  15: return
```

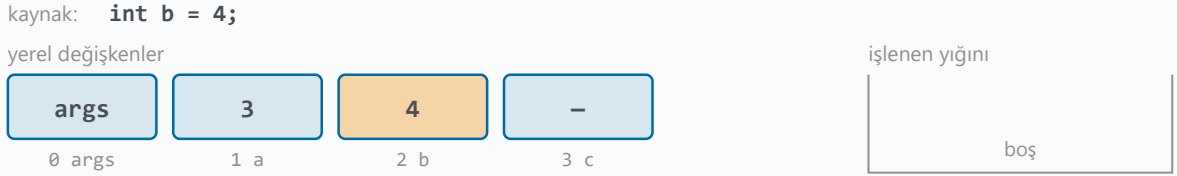
DEĞİŞKENLER

`a = 3`

ÇIKTI

ÇALIŞTIRMA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: JVM Bytecode'u Yürütüyor



Aynı çift b için: 4 yığına, oradan 2 numaralı yuvaya.

Adım 2 / 10

```
 1  0: iconst_3
 2  1: istore_1
 ▶  2: iconst_4
 ▶  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

a = 3 b = 4

ÇIKTI

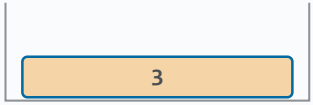
Kod İzleme: JVM Bytecode'u Yürütüyor

kaynak: `int c = a + b;`

yerel değişkenler



işlenen yığını



iload_1 a'nın değerini yığına kopyalar: [3].

Adım 3 / 10

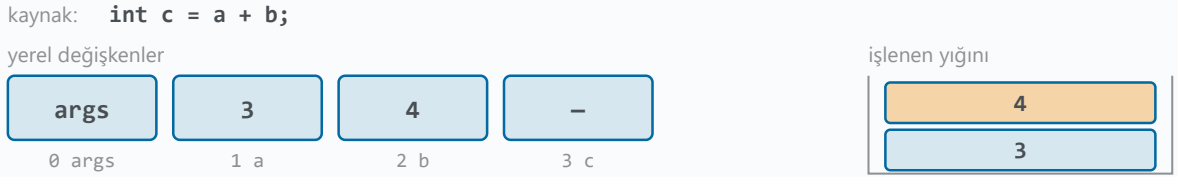
```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 ▶  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

`a = 3` `b = 4`

ÇIKTI

Kod İzleme: JVM Bytecode'u Yürütüyor



iload_2: [3, 4]. Toplanacak iki değer yığının tepesinde.

Adım 4 / 10

```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6▶ 5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

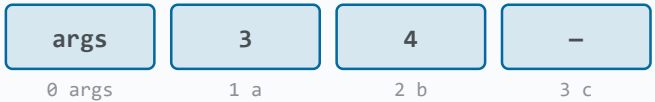
`a = 3` `b = 4`

ÇIKTI

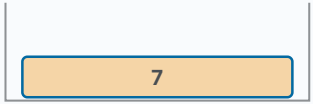
Kod İzleme: JVM Bytecode'u Yürütüyor

kaynak: `int c = a + b;`

yerel değişkenler



işlenen yığını



iadd üstteki iki değeri alır, toplar ve sonucu geri koyar: [7].

Adım 5 / 10

```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

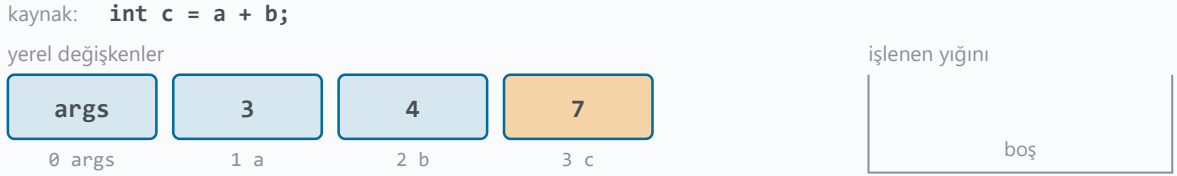
DEĞİŞKENLER

`a = 3` `b = 4`

ÇIKTI

ÇALIŞTIRMA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: JVM Bytecode'u Yürütüyor



istore_3: 7 yığından alınır, c'ye yazılır. Yığın yeniden boş.

Adım 6 / 10

```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 ▶  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

a = 3 | b = 4 | c = 7

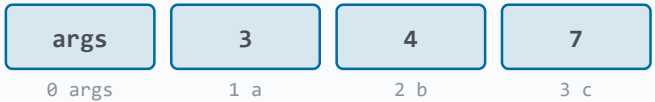
ÇIKTI

ÇALIŞTIRMA · KODU ADIM ADIM ÇALIŞTIR

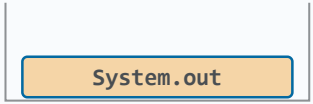
Kod İzleme: JVM Bytecode'u Yürütüyor

kaynak: `System.out.println(c);`

yerel değişkenler



işlenen yığını



getstatic System.out nesnesinin başvurusunu yığına koyar.

Adım 7 / 10

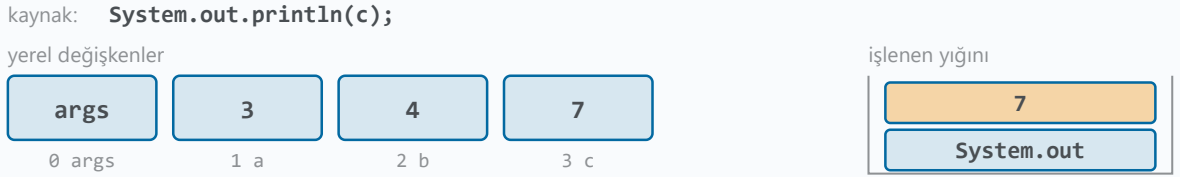
```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 ▶  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

a = 3 b = 4 c = 7

ÇIKTI

Kod İzleme: JVM Bytecode'u Yürütüyor



iload_3: yazdırılacak değer (7) de yığında.

Adım 8 / 10

```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
▶ 11: iload_3
11 12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

a = 3 b = 4 c = 7

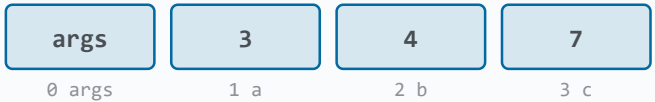
ÇIKTI

ÇALIŞTIRMA · KODU ADIM ADIM ÇALIŞTIR

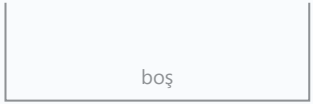
Kod İzleme: JVM Bytecode'u Yürütüyor

kaynak: `System.out.println(c);`

yerel değişkenler



işlenen yığını



invokevirtual iki değeri yığından alır ve println(int)'i çağırır: konsola 7 yazılır.

Adım 9 / 10

```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
▶  12: invokevirtual // println(int)
12 15: return
```

DEĞİŞKENLER

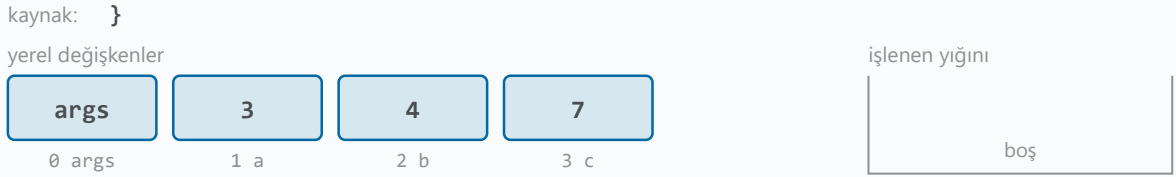
`a = 3` `b = 4` `c = 7`

ÇIKTI

```
7
```

ÇALIŞTIRMA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: JVM Bytecode'u Yürütüyor



return: main biter, çerçeve kaldırılır. Yerel değişkenler ve işlenen yığını çerçeveyle birlikte yok olur.

Adım 10 / 10

```
 1  0: iconst_3
 2  1: istore_1
 3  2: iconst_4
 4  3: istore_2
 5  4: iload_1
 6  5: iload_2
 7  6: iadd
 8  7: istore_3
 9  8: getstatic     // System.out
10 11: iload_3
11 12: invokevirtual // println(int)
▶  15: return
```

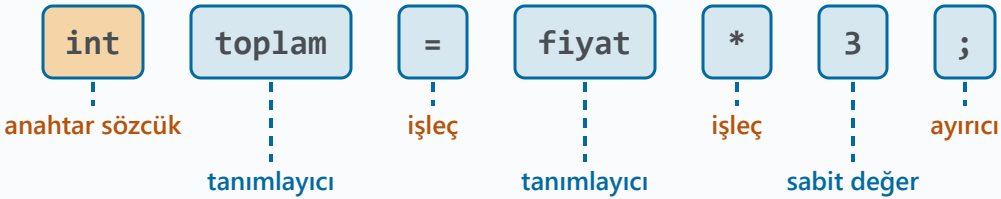
DEĞİŞKENLER

a = 3 b = 4 c = 7

ÇIKTI

```
7
```

Temel Programlama Unsurları



- Derleyici kodu önce **belirteçlere** (token) ayırır: anahtar sözcük, tanımlayıcı, sabit değer, işleç, ayırıcı.
- Java büyük-küçük harfe duyarlıdır: `sayi` ve `Sayi` farklı adlardır.
- Deyimler çoğunlukla noktalı virgülle biter; bloklar süslü parantezle çevrilir.

DİL YAPISI

Boşluk, Girinti ve Süslü Parantez

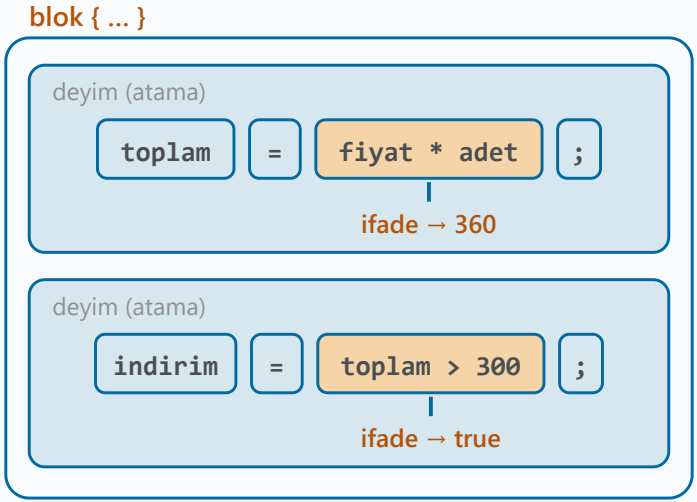
```
public class Tek { public static void main(String[] args) { System.out.println("Merhaba"); } }
```

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

- İki program **aynı bytecode**'a derlenir: derleyici için boşluk, sekme ve satır sonu yalnızca belirteçleri birbirinden ayırır. `x=1;` ile `x = 1;` aynıdır, ama `publicclass` tek bir sözcük sayılır.
- **Girinti** blokların iç içeliğini okuyana gösterir: her yeni blok 4 boşluk içeriden başlar, kapanan `}` açtığı satırın hizasına döner.
- Java geleneğinde açılış `{` satırın sonunda durur. Dizgi içindeki boşluklar ise metnin parçasıdır: `"Merhaba "` ile `"Merhaba"` farklıdır.

İfadeler ve Deyimler

- **İfade** bir değer üretir: `2 + 3` → 5, `sayi > 0` → true.
- **Deyim** bir işlemi tamamlar: `toplam = 5;` veya `System.out.println(toplam);`
- **Blok**, süslü parantez içindeki deyimleri tek yapı olarak gruplar.
- Bir deyimin içinde bir veya daha çok ifade bulunabilir.



İfade, Deyim ve Blok Adım Adım



fiyat için bellekte int kutusu ayrılır ve 120 yazılır.

Adım 1 / 7

```
1 int fiyat = 120;            // tanımlama deyimi
2 int adet = 3;               // tanımlama deyimi
3 int toplam = fiyat * adet; // ifade + atama
4 if (toplam > 300) {         // koşul + blok
5     System.out.println("İndirim");
6 }
```

DEĞİŞKENLER

`fiyat = 120`

ÇIKTI

İfade, Deyim ve Blok Adım Adım



adet kutusu oluşur: 3.

Adım 2 / 7

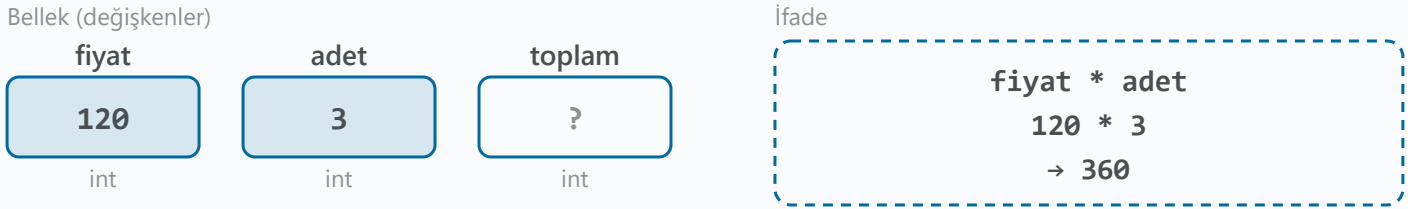
```
1 int fiyat = 120;            // tanımlama deyimi
▶ int adet = 3;               // tanımlama deyimi
3 int toplam = fiyat * adet; // ifade + atama
4 if (toplam > 300) {         // koşul + blok
5     System.out.println("İndirim");
6 }
```

DEĞİŞKENLER

`fiyat = 120` `adet = 3`

ÇIKTI

İfade, Deyim ve Blok Adım Adım



Önce sağdaki **ifade** değerlendirilir: fiyat * adet → 120 * 3 → **360**.

Adım 3 / 7

```
1 int fiyat = 120;            // tanımlama deyimi
2 int adet = 3;               // tanımlama deyimi
▶ int toplam = fiyat * adet; // ifade + atama
4 if (toplam > 300) {         // koşul + blok
5     System.out.println("İndirim");
6 }
```

DEĞİŞKENLER

`fiyat = 120` `adet = 3`

ÇIKTI

İfade, Deyim ve Blok Adım Adım



Üretilen değer **toplam** kutusuna atanır: toplam = 360.

Adım 4 / 7

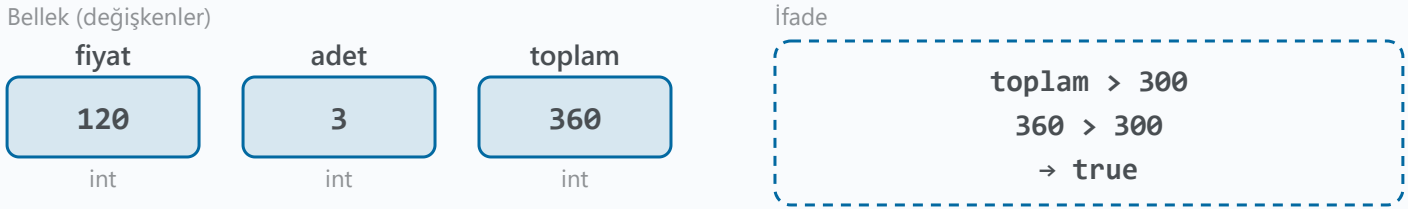
```
1 int fiyat = 120;            // tanımlama deyimi
2 int adet = 3;               // tanımlama deyimi
▶ int toplam = fiyat * adet; // ifade + atama
4 if (toplam > 300) {         // koşul + blok
5     System.out.println("İndirim");
6 }
```

DEĞİŞKENLER

`fiyat = 120` `adet = 3` `toplam = 360`

ÇIKTI

İfade, Deyim ve Blok Adım Adım



Koşul ifadesi: toplam > 300 → 360 > 300 → **true**. Doğru olduğu için bloğa girilir.

Adım 5 / 7

```
1 int fiyat = 120;            // tanımlama deyimi
2 int adet = 3;               // tanımlama deyimi
3 int toplam = fiyat * adet; // ifade + atama
▶ if (toplam > 300) {         // koşul + blok
5     System.out.println("İndirim");
6 }
```

DEĞİŞKENLER

`fiyat = 120` `adet = 3` `toplam = 360`

ÇIKTI

İfade, Deyim ve Blok Adım Adım



Blok içindeki deyim çalışır: konsola **İndirim** yazılır.

Adım 6 / 7

```
1 int fiyat = 120;           // tanımlama deyimi
2 int adet = 3;              // tanımlama deyimi
3 int toplam = fiyat * adet; // ifade + atama
4 if (toplam > 300) {        // koşul + blok
▶     System.out.println("İndirim");
6 }
```

DEĞİŞKENLER

`fiyat = 120` `adet = 3` `toplam = 360`

ÇIKTI

```
İndirim
```

İfade, Deyim ve Blok Adım Adım



Blok biter. adet = 2 olsaydı toplam 240 olur, koşul false olur ve blok hiç çalışmazdı.

Adım 7 / 7

```
1 int fiyat = 120;           // tanımlama deyimi
2 int adet = 3;              // tanımlama deyimi
3 int toplam = fiyat * adet; // ifade + atama
4 if (toplam > 300) {        // koşul + blok
5     System.out.println("İndirim");
▶ }
```

DEĞİŞKENLER

`fiyat = 120` `adet = 3` `toplam = 360`

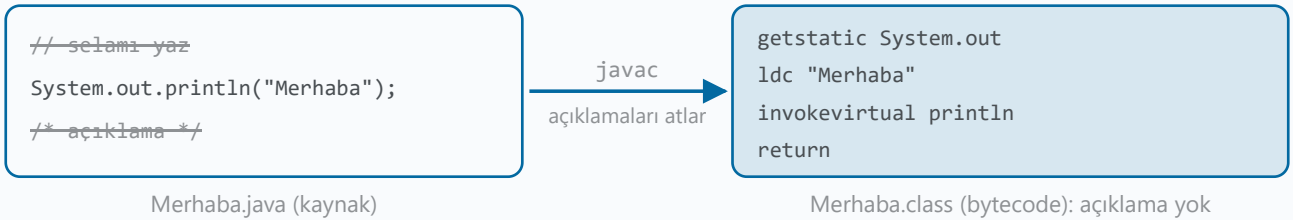
ÇIKTI

```
İndirim
```

Açıklamalar

- `//` satır sonuna kadar açıklamadır.
- `/* ... */` birden fazla satırı kapsar.
- `/** ... */` Javadoc (API belgesi) açıklamasıdır.
- Derleyici açıklamaları atlar; bytecode'a girmezler.
- Açıklama kodun ne yaptığını tekrarlamak yerine **nedenini** açıklamalıdır.

```java
/** Merhaba programı (Javadoc). */
public class Merhaba {
    /* Birden fazla satıra
       yayılan açıklama */
    public static void main(String[] args) {
        // selamı yaz
        System.out.println("Merhaba");
    }
}
```



Merhaba.java (kaynak) — Merhaba.class (bytecode): açıklama yok

KAYNAK KOD

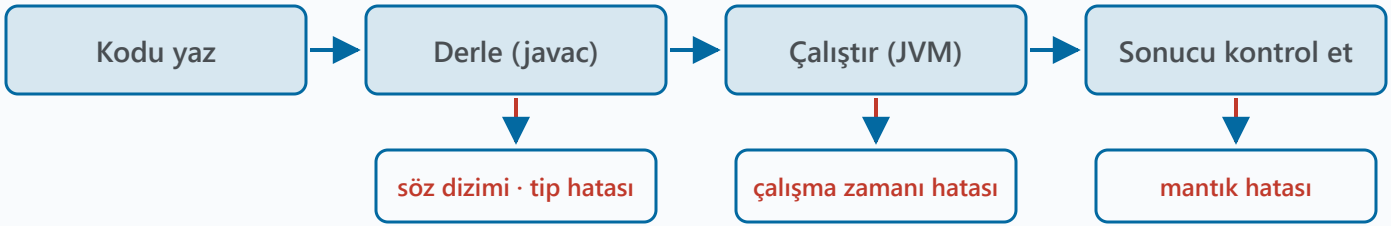
Açıklama Tuzakları

Kod	Ne olur?
`/* dış /* iç */ kalan */`	derleme hatası: `illegal start of expression`. Açıklama **ilk** `*/`'da biter; `kalan */` kod sanılır.
`System.out.println("a // b");`	`a // b` yazılır: tırnak içindeki `//` açıklama değil, metnin parçasıdır.
`// System.out.println(x);`	satır geçici olarak devre dışı kalır; IDE'lerde Ctrl+/ seçili satırları bu biçimde açıklamaya çevirir.
`double fiyat = net * 1.20; // KDV %18`	derlenir, ama açıklama kodla çelişiyor: güncellenmemiş açıklama okuyanı yanıltır.

- İçinde `/* */` bulunan bir bölümü kapatmak için satırları tek tek `//` ile açıklama yapmak daha güvenlidir.
- Açıklama ile kod çeliştiğinde derleyici kodu çalıştırır; doğru olan her zaman koddur.

TEMELLER

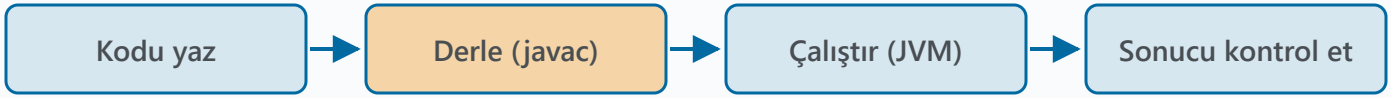
Hata Türleri



Hata	Ne zaman?	Örnek
Söz dizimi	Derleme sırasında	Eksik noktalı virgül
Tip	Derleme sırasında	Metni sayıya atama
Çalışma zamanı	Program çalışırken	Sıfıra bölme
Mantık	Program çalışır ama sonuç yanlıştır	Yanlış koşul

Hata Türleri Örneklerle

Söz dizimi hatası



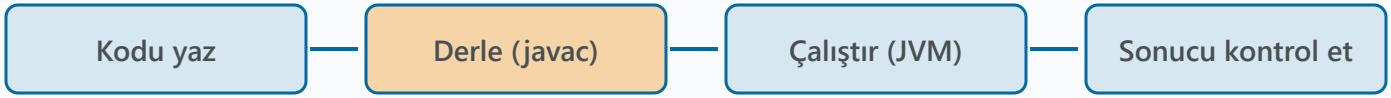

```java
public class Hata1 {
    public static void main(String[] args) {
        System.out.println("Merhaba")
    }
}
```

```
Hata1.java:3: error: ';' expected
        System.out.println("Merhaba")
                                      ^
1 error
```

Derleyici kuralı bozulan yeri satır numarasıyla gösterir; **.class üretilmez**. Düzeltme: satır sonuna `;` ekle.

Hata Türleri Örneklerle

Tip hatası



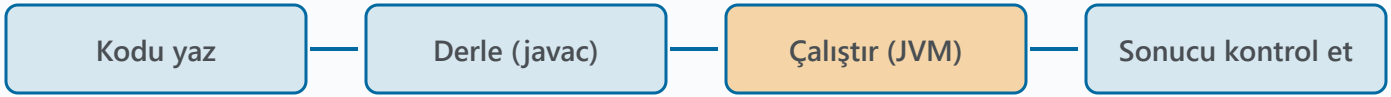

```java
public class Hata2 {
    public static void main(String[] args) {
        int sayi = "42";
    }
}
```

```
Hata2.java:3: error: incompatible types:
        int sayi = "42";
                   ^
1 error
```

Dizgi (String) int kutusuna konamaz. Tipler derleme sırasında denetlenir. Düzeltme: `int sayi = 42;`

Hata Türleri Örneklerle

Çalışma zamanı hatası




```java
public class Hata3 {
    public static void main(String[] args) {
        int a = 10, b = 0;
        System.out.println(a / b);
    }
}
```

```
Exception in thread "main" java.lang.Arit
        at Hata3.main(Hata3.java:4)
```

Kod derlenir, ancak çalışırken tam sayı sıfıra bölünür: JVM bir **istisna** fırlatır ve program durur.

Hata Türleri Örneklerle

Mantık hatası




```java
public class Hata4 {
    public static void main(String[] args) {
        int a = 70, b = 90;
        System.out.println(a + b / 2);
    }
}
```

```
115
```

Hata iletisi yok, ama sonuç yanlış: beklenen 80. `/` önce yapılır: 70 + 45. Düzeltme: `(a + b) / 2`.

Sık Derleme Hataları

Hata	Gerçek ileti	Çözüm
Noktalı virgül yok	`Merhaba.java:3: error: ';' expected`	önceki deyimin sonuna `;` ekle
Metot adı yanlış (`System.out.printn`)	`error: cannot find symbol` `symbol: method printn(String)` `location: variable out of type PrintStream`	adı `println` yap
Küçük harf (`system.out`)	`error: package system does not exist`	`System` büyük S ile yazılır
Ad ile dosya uyuşmuyor	`error: class Selam is public, should be declared in a file named Selam.java`	dosya adı = public sınıf adı
Yanlış çalıştırma (`java Selam`)	`Error: Could not find or load main class Selam`	`.class` yok ya da ad/konum yanlış; uzantı yazma

İletiler JDK 21 ile alınmıştır; sürüme göre ek satırlar (ör. Caused by: ClassNotFoundException) çıkabilir. Her zaman ilk hatadan başla.

Hata İletisini Okumak

```
Not.java:3: error: ';' expected
        int puan = 85
                        ^   ← ^ hatanın fark edildiği yer
1 error
```

dosya satır tür açıklama

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Hata3.main(Hata3.java:4)        ← yığın izi: hangi metot, hangi dosya ve satır
```

- javac iletisi **dosya:satır: error: açıklama** biçimindedir; altında sorunlu kaynak satırı ve hatanın fark edildiği sütunu gösteren ^ işareti yer alır.
- ^ her zaman hatanın kendisini değil, derleyicinin onu **fark ettiği** yeri gösterir: eksik `;` satırın sonunda, eksik `}` ise dosyanın sonunda (`reached end of file while parsing`) bildirilir.
- Çalışma zamanı hatasında ilk satır istisnanın türünü ve ayrıntısını, **at** ile başlayan **yığın izi** satırları hatanın oluştuğu metodu ve satırı verir.

Kod İzleme: Hataları Sırayla Düzeltme

3 `int puan = 85`	✗ bildirildi
4 `System.out.printn("Puan: " + puan);`	? henüz bildirilmedi

Not.class: yok

İlk derleme: yalnız **bir** hata bildirilir. javac söz dizimi bozuk olduğu sürece anlam denetimine geçmez; 4. satırdaki yanlış henüz görünmez.

Adım 1 / 5

```
1 public class Not {
2     public static void main(String[] args) {
▶           int puan = 85
4           System.out.printn("Puan: " + puan);
5     }
6 }
```

DEĞİŞKENLER

`derleme = başarısız` `hata sayısı = 1`

ÇIKTI

```
> javac Not.java
Not.java:3: error: ';' expected
        int puan = 85
                   ^
1 error
```

Kod İzleme: Hataları Sırayla Düzeltme

```
3  int puan = 85;                          ✓ düzeltildi
4  System.out.printn("Puan: " + puan);     ? henüz bildirilmedi
```

Not.class: yok

İletinin gösterdiği yer düzeltilir: satır sonuna `;` eklenir ve dosya kaydedilir.

Adım 2 / 5

```
1 public class Not {
2     public static void main(String[] args) {
▶         int puan = 85
4         System.out.printn("Puan: " + puan);
5     }
6 }
```

DEĞİŞKENLER

`derleme = yapılmadı` `hata sayısı = ?`

ÇIKTI

```
> javac Not.java
Not.java:3: error: ';' expected
        int puan = 85
                   ^
1 error
```

Kod İzleme: Hataları Sırayla Düzeltme

```
3  int puan = 85;
```
✓ düzeltildi

```
4  System.out.printn("Puan: " + puan);
```
✗ şimdi bildirildi

Not.class: yok

Yeniden derleme: söz dizimi artık doğru, anlam denetimi çalışır ve **printn** adlı metodu bulamaz.

Adım 3 / 5

```
1 public class Not {
2     public static void main(String[] args) {
3         int puan = 85
▶         System.out.printn("Puan: " + puan);
5     }
6 }
```

DEĞİŞKENLER

derleme = **başarısız** | hata sayısı = **1**

ÇIKTI

```
> javac Not.java
Not.java:4: error: cannot find symbol
        System.out.printn("Puan: " + puan);
                   ^
  symbol:   method printn(String)
  location: variable out of type PrintStream
1 error
```

Kod İzleme: Hataları Sırayla Düzeltme

```
3  int puan = 85;                                  ✓ düzeltildi
4  System.out.println("Puan: " + puan);           ✓ düzeltildi
```

Not.class: yok

symbol: satırı bulunamayan adı söyler: metot adı `println` olarak düzeltilir.

Adım 4 / 5

```
1 public class Not {
2     public static void main(String[] args) {
3         int puan = 85
▶         System.out.printn("Puan: " + puan);
5     }
6 }
```

DEĞİŞKENLER

`derleme = yapılmadı` `hata sayısı = ?`

ÇIKTI

```
> javac Not.java
Not.java:4: error: cannot find symbol
        System.out.printn("Puan: " + puan);
                   ^
  symbol:   method printn(String)
  location: variable out of type PrintStream
1 error
```

Kod İzleme: Hataları Sırayla Düzeltme

```
3  int puan = 85;                                  ✓ düzeltildi
4  System.out.println("Puan: " + puan);           ✓ düzeltildi
```

Not.class: oluştu

Üçüncü derleme sessizce başarılı olur, Not.class oluşur ve program çalışır. Hata sayısı bir düzeltmeden sonra azalmayabilir, artabilir de: her zaman ilk hatadan başlanır.

Adım 5 / 5

```
1 public class Not {
2     public static void main(String[] args) {
3         int puan = 85
▶           System.out.printn("Puan: " + puan);
5     }
6 }
```

DEĞİŞKENLER

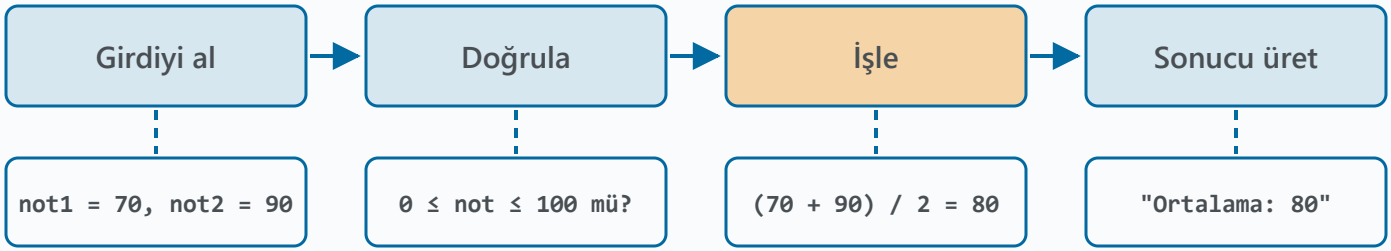
`derleme = başarılı` `hata sayısı = 0`

ÇIKTI

```
> javac Not.java
> java Not
Puan: 85
```

TASARIM

Program Bir Tariftir



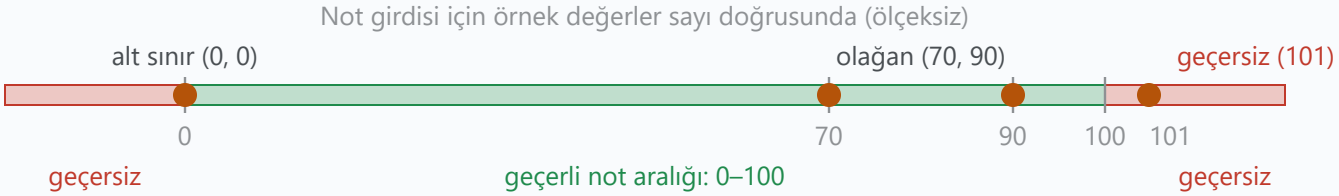
- Her adımın ön koşulu ve ürettiği değer belirlenmelidir.
- Büyük tarifler alt görevlere ayrılır (Java'da bunlar ileride **metot** olacak).

Tarifi kodlamadan önce elle sınayın: 70 ve 90 için adımları izleyince 80 çıkıyor mu? 101 girilirse hangi adım durdurur?

TASARIM

Kodlamadan Önce

- Örnek girdi ve beklenen çıktıyı yazın.
- Çözümü doğal dil veya sözde kodla adımlara ayırın.
- Sınır durumlarını ve hata davranışını belirleyin.
- Ardından adımları Java yapılarına dönüştürün.

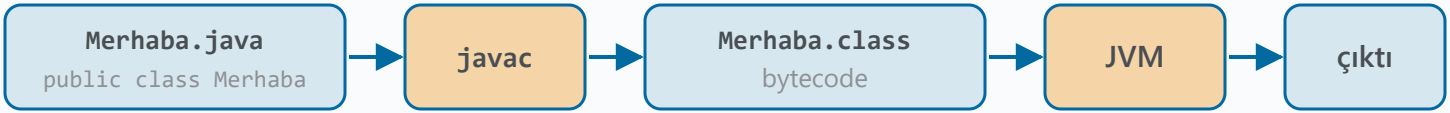


Örnek girdi	Beklenen çıktı	Neden?
70, 90	80	olağan durum
0, 0	0	alt sınır
101, 50	hata iletisi	geçersiz girdi

ÖZET

Bölüm Özeti

- Program, bilgisayarın bir görevi yerine getirmek için yürüttüğü komutlar bütünüdür.
- Java kaynak dosyası javac ile bytecode'a derlenir ve JVM üzerinde çalışır.
- public sınıf adı dosya adıyla eşleşir; main programın giriş noktasıdır.
- İfadeler değer üretir, deyimler işlemi tamamlar, bloklar deyimleri gruplar.
- Kodlamadan önce örnek girdi, beklenen çıktı ve sınır durumları belirlenir.



Dosya adı = public sınıf adı; JVM çalıştırmaya main metodundan başlar.

ÖZET

Kontrol Soruları

1. JDK, bytecode ve JVM arasındaki ilişki nedir?
2. Merhaba.java dosyası hangi komutlarla derlenip çalıştırılır?
3. main metodunun görevi nedir?
4. İfade ile deyim arasındaki fark nedir?
5. Söz dizimi hatası ile mantık hatası nasıl ayrılır?
6. Kaynak kod değiştirilip yeniden derlenmezse `java` komutu neyi çalıştırır?
7. `"Puan: " + 40 + 45` ifadesi neden `Puan: 4045` verir?
8. Komut satırı argümanları programa nasıl ulaşır?

ÖZET

Alıştırmalar

1. Adınızı, bölümünüzü ve öğrenci numaranızı üç ayrı satırda yazdıran `Kimlik` programını yazın, derleyin, çalıştırın.
2. Aynı çıktıyı tek bir `println` ve `\n` kullanarak üretin.
3. `\t` ile üç satırlık bir ders–kredi tablosu yazdırın.
4. `System.out.println("Merhaba")` satırındaki `;` işaretini silip derleyin; hata iletisinin satır numarasını ve ^ işaretini yorumlayın.
5. Sınıf adını `Merhaba`, dosya adını `merhaba.java` yapıp derleyin. Ne olur, neden?
6. `System.out.println(5 + 3);` ile `System.out.println("5 + 3");` çıktılarını tahmin edin; çalıştırarak doğrulayın.

ÖZET

Kaynaklar

- Deitel ve Deitel, Java: How to Program, Early Objects, 11. baskı, Pearson, 2018.
- Liang, Introduction to Java Programming and Data Structures, 13. baskı, Pearson, 2024.
- Oracle, The Java Tutorials: **A Closer Look at the "Hello World!" Application**.
- Oracle, **The Java Language Specification, Java SE 21 Edition**.
- Sedgewick ve Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **1.1 Your First Program**.
- Downey ve Mayfield, Think Java, 2. baskı, O'Reilly, 2019 (**ücretsiz çevrim içi sürüm**).

Son

Java'ya Giriş ve İlk Program

Sonraki sunum: Değişkenler, Veri Tipleri ve İşleçler · Çalışma soruları · Simülatör: Java Derleme ve Çalıştırma · Terim sözlüğü