Bölüm 6: İstisnalar

Java ile Nesne Yönelimli Programlama

İstisna kavramı · try ve catch · Çoklu catch · finally · try-with-resources · İç içe try · İstisna hiyerarşisi · Kontrol edilen / edilmeyen · throw ve throws · Yaygın istisna sınıfları · Error · Özel istisnalar · Yığın izi

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/java/slides/Bolum_06_Istisnalar.html

GİRİŞ

Öğrenme Çıktıları

Kavram

İstisnayı (exception) tanımlamak; çalışma zamanı sorunlarını `Exception` ve `Error` olarak ayırt etmek.

Yakalama

`try`, `catch`, `finally` ve try-with-resources ile istisnaları ele almak; akışı adım adım izlemek.

Hiyerarşi

`Throwable`, `Error`, `Exception`, `RuntimeException` ilişkisini ve catch sırasının önemini açıklamak.

Türler

Kontrol edilen (checked) ve kontrol edilmeyen (unchecked) istisnaları ayırmak; "ele al veya bildir" kuralını uygulamak.

Fırlatma

`throw` ile istisna fırlatmak, `throws` ile bildirmek; istisnanın çağrı yığınında yayılımını izlemek.

Uygulama

Yaygın istisna sınıflarını tanımak, özel istisna sınıfı yazmak ve yığın izini (stack trace) okumak.

İSTİSNA KAVRAMI

İstisna Nedir?

- **İstisna (exception)**: program çalışırken ortaya çıkan, programın **normal akışını bozan** beklenmeyen bir durumdur.
- Örnek: açılmak istenen dosyanın bulunamaması, sıfıra bölme, dizinin dışına erişim, `null` bir referansın kullanılması.
- İstisnalar Java çalışma zamanı sistemi (JVM) tarafından **otomatik** olarak ya da programcı tarafından kodla (`throw`) oluşturulabilir.
- Java'da istisna bir **nesnedir**: tipini, mesajını ve nerede oluştuğunu taşır.
- İstisna mekanizması, programcıya anormal durumlarla **düzenli** bir şekilde başa çıkma şansı verir.

Java bir istisnayla karşılaştığında kabaca şunu der: **"Pes ettim, şu an ne yapacağımı bilmiyorum. Bunu biri halletsin!"**

Benzetme

Garson mutfakta yemeğin bittiğini öğrenir. Kendisi çözemez; durumu **yöneticiye bildirir**. Yönetici ya çözer (başka yemek önerir) ya da durumu bir üste iletir.

İSTİSNA KAVRAMI

Çalışma Zamanı Hataları: İstisna ve Error

Derleme hataları programı hiç çalıştırmaz, mantık hataları ise hiçbir sinyal üretmez. Program **çalışırken** ortaya çıkan sorunu JVM bir `Throwable` nesnesi olarak fırlatır; bu nesnenin iki ana kolu vardır.

Kol	Ne anlatır?	Program ne yapabilir?	Örnek
İstisna (`Exception`)	Programın öngörebileceği, olağan dışı bir durum	Yakalayıp toparlanabilir	`10 / 0`, `dizi[5]` (3 öğeli dizide), olmayan dosya
Hata (`Error`)	JVM ya da sistem düzeyinde ciddi bir sorun	Genellikle yakalanmaz; program sonlanır	`StackOverflowError`, `OutOfMemoryError`

İstisna mekanizması ve yığın izi, çalışma zamanı hatalarının nedenini bulup düzeltmenin (hata ayıklama, debugging) en güçlü araçlarındandır.

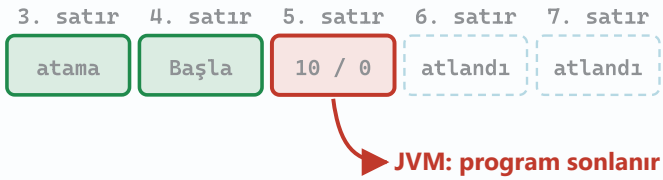
Yakalanmayan İstisna: Program Çöker

```
public class Cokus {
  public static void main(String[] args) {
    int pay = 10, payda = 0;
    System.out.println("Başla");
    int sonuc = pay / payda;
    System.out.println("Sonuç: " + sonuc);
    System.out.println("Bitti");
  }
}
```

Çıktı

```
Başla
Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Cokus.main(Cokus.java:5)
```

- 5. satırda tam sayı sıfıra bölünür; JVM bir `ArithmeticException` nesnesi **fırlatır**.
- Kimse yakalamadığı için program **sonlanır**: "Sonuç" ve "Bitti" hiç yazılmaz.
- JVM, istisnanın **tipini**, **mesajını** ve **yerini** (sınıf, metot, satır) yazdırır.



İSTİSNA KAVRAMI

İstisnayla Başa Çıkmanın İki Yolu

1) Ele almak (handle)

Metot, istisna oluşturabilecek kodu `try` içine alır ve `catch` ile durumu kendisi çözer.
Örnek: dosya okuma metodu, dosya bulunamazsa (`FileNotFoundException`) kullanıcıya mesaj gösterip varsayılan ayarlarla devam eder.

2) Çağıranın sorunu yapmak (declare)

Metot istisnayı ele almaz; imzasında `throws` ile bildirir. İstisna, metodu **çağıran kodun** sorunu olur.
Çağıran kod da aynı seçimle karşılaşır: ya ele alır ya da bir üste iletir.

Hangi yol? İstisnayı **anlamlı biçimde çözebilecek** yer ele almalıdır. Örneğin dosya adını kullanıcıdan alan kod, "dosya yok" durumunda yeniden sorabilir; düşük düzeyli okuma metodu ise bunu bilemez.

"Ters Giden Bir Şey": Dönüş Kodu ile

```java
public static int indexOf(String[] isimler, String isim) {
  for (int i = 0; i < isimler.length; i++) {
    if (isimler[i].equals(isim)) {
      return i;
    }
  }
  return -1;
}
// main içinde
String[] isimler = {"Ali", "Ayşe", "Can"};
System.out.println(indexOf(isimler, "Can"));
int k = indexOf(isimler, "Ece");   // k = -1
System.out.println(isimler[k]);
```

Çıktı

```
2
java.lang.ArrayIndexOutOfBoundsException:
  Index -1 out of bounds for length 3
```

- İstisnalar, bir programın **"bir şeyin ters gittiğini"** bildirmesi için kullanılır.
- Ama "ters gitmek" **özneldir**: aranan ismin bulunamaması hata mıdır, olağan bir sonuç mu?
- Bu metot, eşleşme bulamazsa istisna fırlatmak yerine `-1` döndürür.
- Çağıran, `-1`'i kontrol etmeyi **unutursa** hata başka bir yerde, `isimler[-1]` erişiminde ortaya çıkar (soldaki çıktı).

Dönüş Kodu Yerine İstisna

```java
public static int bul(String[] isimler, String isim) {
  for (int i = 0; i < isimler.length; i++) {
    if (isimler[i].equals(isim)) {
      return i;
    }
  }
  throw new IllegalArgumentException("Bulunamadı: " + isim);
}
// main içinde
try {
  System.out.println(bul(isimler, "Ece"));
} catch (IllegalArgumentException e) {
  System.out.println("Hata: " + e.getMessage());
}
```

Çıktı

```
Hata: Bulunamadı: Ece
```

- Genellikle `-1` gibi **özel dönüş kodlarından** kaçınılmalı, Java'nın istisna mekanizması tercih edilmelidir.
- İstisna, hata durumunu **açık** ve **etkili** biçimde ele almanın yoludur.

İstisnaların yararı

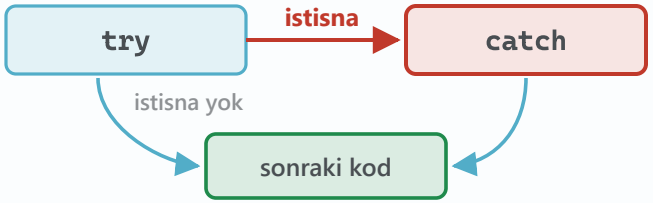
+ Açık ve tutarlı hata işleme; hata normal sonuçla karışmaz
+ Okunabilirlik artar, hata ayıklama kolaylaşır
+ Kodun güvenilirliği artar

try ve catch Deyimleri

```java
try {
  // Potansiyel hata içeren kodlar
  int sonuc = 10 / 0; // Hata: ArithmeticException
} catch (ArithmeticException e) {
  // Hata durumuyla başa çıkma
  System.out.println("hata oluştu: " + e.getMessage());
}
```

Çıktı

```
hata oluştu: / by zero
```



- Hata yönetiminin **temel yapısıdır**.
- `try` bloğu: **potansiyel hata** içeren kodlar.
- `catch` bloğu: `try` içinde belirtilen tipte bir istisna oluşursa çalışır ve onu **ele alır**.
- `catch (ArithmeticException e)` : `e` , fırlatılan istisna **nesnesine** referanstır (bir parametre gibi).
- İstisna oluşursa `try` 'ın **kalan satırları atlanır**; istisna oluşmazsa `catch` hiç çalışmaz.
- catch'ten sonra program **normal akışına** devam eder.

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
▶      int pay = 10, payda = 0;
4      System.out.println("Başla");
5      try {
6        System.out.println("Bölme öncesi");
7        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
9      } catch (ArithmeticException e) {
10       System.out.println("Hata: " +
e.getMessage());
11     }
12     System.out.println("Bitti");
13   }
14 }
```

```
println("Bölme öncesi")
sonuc = pay / payda
println("Sonuç: " + …)
catch (ArithmeticException e)
println("Hata: " + …)
println("Bitti")
```

Değişkenler oluşur. `payda` sıfır: sorun yaklaşıyor.

Adım 1 / 7

ÇIKTI

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
3      int pay = 10, payda = 0;
▶      System.out.println("Başla");
5      try {
6        System.out.println("Bölme öncesi");
7        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
9      } catch (ArithmeticException e) {
10       System.out.println("Hata: " +
e.getMessage());
11     }
12     System.out.println("Bitti");
13   }
14 }
```

```
println("Bölme öncesi")
sonuc = pay / payda
println("Sonuç: " + …)
catch (ArithmeticException e)
println("Hata: " + …)
println("Bitti")
```

`try` dışındaki normal kod çalışır.

Adım 2 / 7

ÇIKTI

```
Başla
```

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
3      int pay = 10, payda = 0;
4      System.out.println("Başla");
5      try {
▶        System.out.println("Bölme öncesi");
7        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
9      } catch (ArithmeticException e) {
10       System.out.println("Hata: " +
e.getMessage());
11     }
12     System.out.println("Bitti");
13   }
14 }
```

```
println("Bölme öncesi")
sonuc = pay / payda
println("Sonuç: " + …)
catch (ArithmeticException e)
println("Hata: " + …)
println("Bitti")
```

`try` bloğuna girildi; ilk satır sorunsuz çalışır.

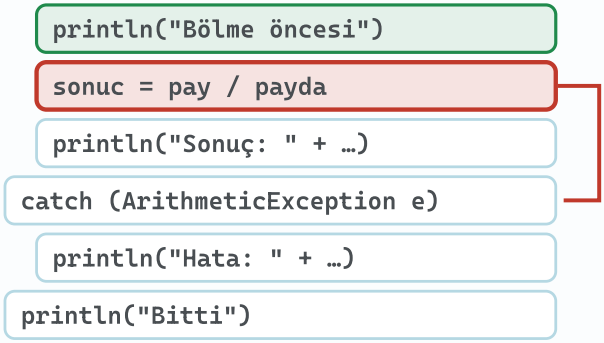
Adım 3 / 7

ÇIKTI

```
Başla
Bölme öncesi
```

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
3      int pay = 10, payda = 0;
4      System.out.println("Başla");
5      try {
6        System.out.println("Bölme öncesi");
▶        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
9      } catch (ArithmeticException e) {
10       System.out.println("Hata: " +
   e.getMessage());
11     }
12     System.out.println("Bitti");
13   }
14 }
```



10 / 0: JVM bir `ArithmeticException` nesnesi oluşturur ve **fırlatır**. Atama yapılmaz.

Adım 4 / 7

ÇIKTI

```
Başla
Bölme öncesi
```

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
3      int pay = 10, payda = 0;
4      System.out.println("Başla");
5      try {
6        System.out.println("Bölme öncesi");
7        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
▶      } catch (ArithmeticException e) {
10       System.out.println("Hata: " +
e.getMessage());
11     }
12     System.out.println("Bitti");
13   }
14 }
```

println("Bölme öncesi")

sonuc = pay / payda

println("Sonuç: " + …)

catch (ArithmeticException e) ✓

println("Hata: " + …)

println("Bitti")

Tip eşleşir: `catch` çalışır, `e` istisna nesnesini gösterir. “Sonuç” satırı **atlandı**.

Adım 5 / 7

ÇIKTI

```
Başla
Bölme öncesi
```

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
3      int pay = 10, payda = 0;
4      System.out.println("Başla");
5      try {
6        System.out.println("Bölme öncesi");
7        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
9      } catch (ArithmeticException e) {
▶        System.out.println("Hata: " +
e.getMessage());
11     }
12     System.out.println("Bitti");
13   }
14 }
```

println("Bölme öncesi")

sonuc = pay / payda

println("Sonuç: " + …)

catch (ArithmeticException e)

println("Hata: " + …)

println("Bitti")

e.getMessage() → "/ by zero".

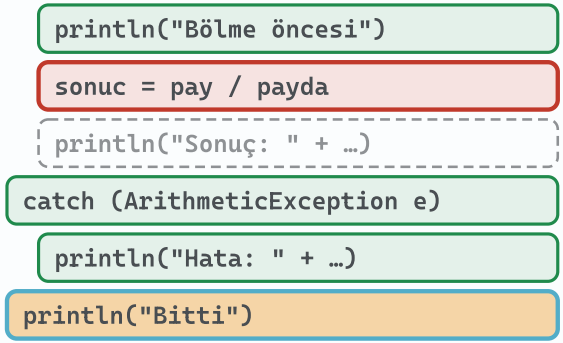
Adım 6 / 7

ÇIKTI

```
Başla
Bölme öncesi
Hata: / by zero
```

try-catch Akışı: Adım Adım

```
1  public class BolmeOrnegi {
2    public static void main(String[] args) {
3      int pay = 10, payda = 0;
4      System.out.println("Başla");
5      try {
6        System.out.println("Bölme öncesi");
7        int sonuc = pay / payda;
8        System.out.println("Sonuç: " + sonuc);
9      } catch (ArithmeticException e) {
10       System.out.println("Hata: " +
   e.getMessage());
11     }
▶      System.out.println("Bitti");
13   }
14 }
```



Program **çökmedi**: catch'ten sonra normal akış devam eder.

Adım 7 / 7

ÇIKTI

```
Başla
Bölme öncesi
Hata: / by zero
Bitti
```

İstisna Nesnesinden Bilgi Almak

```java
try {
  int[] dizi = new int[3];
  dizi[5] = 7;
} catch (ArrayIndexOutOfBoundsException e) {
  System.out.println(e.getMessage());
  System.out.println(e);
  System.out.println(e.getClass().getSimpleName());
  e.printStackTrace();
}
```

Çıktı

```
Index 5 out of bounds for length 3
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
ArrayIndexOutOfBoundsException
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
    at IstisnaNesnesi.main(IstisnaNesnesi.java:5)
```

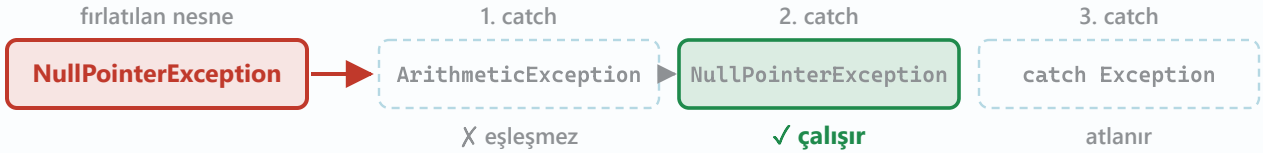
Metot	Ne verir?
`getMessage()`	Yalnızca açıklama mesajı (yoksa `null`)
`toString()`	Tam sınıf adı + ": " + mesaj; `println(e)` bunu kullanır
`getClass()`	İstisnanın gerçek sınıfı
`printStackTrace()`	`toString()` + yığın izi; **System.err**'e yazar
`getCause()`	Bu istisnaya yol açan başka istisna (varsa)

TRY VE CATCH

Çoklu catch Bloğu

```
try {
  // Potansiyel hata içeren kodlar
} catch (ArithmeticException e) {
  // ArithmeticException ile başa çıkma
} catch (NullPointerException e) {
  // NullPointerException ile başa çıkma
} catch (Exception e) {
  // Diğer hata türleri için genel durum
}
```

- Bir kod bloğunda **birden çok** tipte istisna oluşabilir.
- Her hata türü için ayrı bir `catch` bloğu eklenir; her blok o tipe **özgü** tepkiyi içerir.
- İstisna fırlatıldığında catch blokları **yukarıdan aşağıya** sırayla denenir; **eşleşen ilk** blok çalışır, diğerleri atlanır.
- "Eşleşmek": fırlatılan nesne, catch'teki tipin **kendisi veya alt sınıfıdır** (`instanceof` mantığı).
- Son sıradaki `catch (Exception e)` geri kalan tüm istisnaları yakalayan **genel** durumdur.



TRY VE CATCH

Zincirleme catch Blokları: Sıra Önemlidir

✗ GENEL TİP ÖNCE

```
try {
  int sonuc = 10 / 0;
} catch (Exception e) {
  System.out.println("Genel hata");
} catch (ArithmeticException e) {
  System.out.println("Aritmetik hata");
}
```

```
SiraHatasi.java:7: error: exception
ArithmeticException
    has already been caught
```

- catch tipleri arasında **kalıtım ilişkisi** varsa sıralama önemlidir: **alt sınıflar, üst sınıflardan önce** gelmelidir.
- `ArithmeticException` bir `Exception` 'dır; ilk blok onu zaten yakalar, ikinci blok **asla çalışamaz**. Derleyici buna izin vermez.
- Kalıtım ilişkisi olmayan tiplerin (`ArithmeticException`, `NullPointerException`) sırası önemli değildir.

✓ ÖZELDEN GENELE

```
} catch (ArithmeticException e) {
  System.out.println("Aritmetik hata");
} catch (Exception e) {
  System.out.println("Genel hata");
}
```

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
▶      int y = Integer.parseInt(girdi[1]);
5      System.out.println("Sonuç: " + (x / y));
6    } catch (NumberFormatException e) {
7      System.out.println("Sayı değil: " +
   e.getMessage());
8    } catch (ArithmeticException e) {
9      System.out.println("Sıfır: " + e.getMessage());
10   } catch (ArrayIndexOutOfBoundsException e) {
11     System.out.println("Eksik: " + e.getMessage());
12   } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

`NumberFormatException`

1. `catch: NumberFormat`
2. `catch: Arithmetic`
3. `catch: ArrayIndexOutOfBounds`
4. `finally`

1. çağrı: `parseInt("abc")` → `NumberFormatException`. Bölme satırı atlanır.

Adım 1 / 7

ÇIKTI

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
4      int y = Integer.parseInt(girdi[1]);
5      System.out.println("Sonuç: " + (x / y));
▶    } catch (NumberFormatException e) {
▶      System.out.println("Sayı değil: " +
   e.getMessage());
8    } catch (ArithmeticException e) {
9      System.out.println("Sıfır: " + e.getMessage());
10   } catch (ArrayIndexOutOfBoundsException e) {
11     System.out.println("Eksik: " + e.getMessage());
12   } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

NumberFormatException

1. catch: NumberFormat ✓

2. catch: Arithmetic

3. catch: ArrayIndexOutOfBounds

finally

İlk catch eşleşir ve çalışır; diğer catch'lere **bakılmaz**.

Adım 2 / 7

ÇIKTI

```
Sayı değil: For input string: "abc"
```

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
4      int y = Integer.parseInt(girdi[1]);
5      System.out.println("Sonuç: " + (x / y));
6    } catch (NumberFormatException e) {
7      System.out.println("Sayı değil: " +
   e.getMessage());
8    } catch (ArithmeticException e) {
9      System.out.println("Sıfır: " + e.getMessage());
10   } catch (ArrayIndexOutOfBoundsException e) {
11     System.out.println("Eksik: " + e.getMessage());
▶    } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

`NumberFormatException`

- `1. catch: NumberFormat` ✓
- `2. catch: Arithmetic`
- `3. catch: ArrayIndexOutOfBounds`
- `finally`

Ardından `finally`.

Adım 3 / 7

ÇIKTI

```
Sayı değil: For input string: "abc"
finally çalıştı
```

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
4      int y = Integer.parseInt(girdi[1]);
▶      System.out.println("Sonuç: " + (x / y));
6    } catch (NumberFormatException e) {
7      System.out.println("Sayı değil: " +
   e.getMessage());
8    } catch (ArithmeticException e) {
9      System.out.println("Sıfır: " + e.getMessage());
10   } catch (ArrayIndexOutOfBoundsException e) {
11     System.out.println("Eksik: " + e.getMessage());
12   } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

`ArithmeticException`

1. `catch: NumberFormat`
2. `catch: Arithmetic`
3. `catch: ArrayIndexOutOfBounds`
4. `finally`

2. çağrı: `10 / 0` → `ArithmeticException`.

Adım 4 / 7

ÇIKTI

```
Sayı değil: For input string: "abc"
finally çalıştı
```

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
4      int y = Integer.parseInt(girdi[1]);
5      System.out.println("Sonuç: " + (x / y));
6    } catch (NumberFormatException e) {
7      System.out.println("Sayı değil: " +
   e.getMessage());
▶    } catch (ArithmeticException e) {
▶      System.out.println("Sıfır: " + e.getMessage());
10   } catch (ArrayIndexOutOfBoundsException e) {
11     System.out.println("Eksik: " + e.getMessage());
12   } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

ArithmeticException

1. catch: NumberFormat ✗

2. catch: Arithmetic ✓

3. catch: ArrayIndexOutOfBounds

finally

1. catch eşleşmez (✗), 2. catch eşleşir (✓).

Adım 5 / 7

ÇIKTI

```
Sayı değil: For input string: "abc"
finally çalıştı
Sıfır: / by zero
```

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
▶      int y = Integer.parseInt(girdi[1]);
5      System.out.println("Sonuç: " + (x / y));
6    } catch (NumberFormatException e) {
7      System.out.println("Sayı değil: " +
   e.getMessage());
8    } catch (ArithmeticException e) {
9      System.out.println("Sıfır: " + e.getMessage());
10   } catch (ArrayIndexOutOfBoundsException e) {
11     System.out.println("Eksik: " + e.getMessage());
12   } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

`ArrayIndexOutOfBoundsException`

`1. catch: NumberFormat`

`2. catch: Arithmetic`

`3. catch: ArrayIndexOutOfBounds`

`finally`

3. çağrı: dizinin tek öğesi var; `girdi[1]` → `ArrayIndexOutOfBoundsException`.

Adım 6 / 7

ÇIKTI

```
Sayı değil: For input string: "abc"
finally çalıştı
Sıfır: / by zero
finally çalıştı
```

Hangi catch Çalışır?

```
1  static void bol(String[] girdi) {
2    try {
3      int x = Integer.parseInt(girdi[0]);
4      int y = Integer.parseInt(girdi[1]);
5      System.out.println("Sonuç: " + (x / y));
6    } catch (NumberFormatException e) {
7      System.out.println("Sayı değil: " +
   e.getMessage());
8    } catch (ArithmeticException e) {
9      System.out.println("Sıfır: " + e.getMessage());
▶    } catch (ArrayIndexOutOfBoundsException e) {
▶      System.out.println("Eksik: " + e.getMessage());
▶    } finally {
13     System.out.println("finally çalıştı");
14   }
15 }
16 // main içinde
17 bol(new String[] {"10", "abc"});
18 bol(new String[] {"10", "0"});
19 bol(new String[] {"10"});
```

`ArrayIndexOutOfBoundsException`

- `1. catch: NumberFormat` ✗
- `2. catch: Arithmetic` ✗
- `3. catch: ArrayIndexOutOfBounds` ✓
- `finally`

İlk iki catch eşleşmez, üçüncüsü eşleşir; sonra `finally`.

Adım 7 / 7

ÇIKTI

```
Sayı değil: For input string: "abc"
finally çalıştı
Sıfır: / by zero
finally çalıştı
Eksik: Index 1 out of bounds for
length 1
finally çalıştı
```

Tek catch, Birden Çok Tip (Java 7+)

```
try {
  Object o = "Merhaba";
  Integer i = (Integer) o;
} catch (ClassCastException
    | NullPointerException e) {
  System.out.println("Tür veya null hatası: "
    + e.getClass().getSimpleName());
}
```

Çıktı

```
Tür veya null hatası: ClassCastException
```

- Tepki aynıysa tipler `|` ile birleştirilir; kod tekrarı azalır.
- Bu blokta `e` **final** sayılır; yeni değer atanamaz.

✗ ALT VE ÜST SINIF BİRLİKTE YAZILAMAZ

```
} catch (FileNotFoundException
    | IOException e) {
```

```
error: Alternatives in a multi-catch
statement
  cannot be related by subclassing
  Alternative FileNotFoundException is a
  subclass of alternative IOException
```

Çözüm: yalnızca üst tip yazılır: `catch (IOException e)` . Alt tip zaten onun içindedir.

FINALLY

finally Bloğu

```
try {
  // Potansiyel hata içeren kodlar
} catch (Exception e) {
  // Diğer hata türleri için genel durum
} finally {
  // Her durumda çalıştırılacak kodlar
}
```

- `try` ve `catch` bloklarının ardından eklenir.
- İstisna **olsun veya olmasın**, yakalansın veya yakalanmasın, **her durumda** çalışır.
- `try` veya `catch` içinde `return` olsa bile, metottan çıkmadan **önce** çalışır.
- Kodun belirli adımlarının çalışmasını **güvence altına** alır: dosya, ağ ve veri tabanı bağlantılarını kapatmak gibi **kaynak yönetimi** ve temizlik işleri.
- `try` 'dan sonra en az bir `catch` **veya** `finally` gelmelidir; `try { } finally { }` geçerlidir.
- Asıl istisna: `System.exit()` JVM'i hemen durdurur; finally çalışmaz.

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
▶    System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

dönüş değeri henüz yok

`println` 'dan önce argümanı hesaplanmalı: `bol(10, 2)` çağrılır.

Adım 1 / 13

ÇAĞRI YIĞINI

`main()`

ÇIKTI

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
▶      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

try { … }

catch (ArithmeticException e)

finally { … }

dönüş değeri henüz yok

try bloğu başlar.

Adım 2 / 13

ÇAĞRI YIĞINI

bol(10, 2)

main()

ÇIKTI

```
try başladı
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
▶      int s = a / b;
▶      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

dönüş değeri henüz yok

Bölme başarılı: `s = 5`.

Adım 3 / 13

ÇAĞRI YIĞINI

`bol(10, 2)`

`main()`

ÇIKTI

```
try başladı
try bitti
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
▶      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

bekleyen dönüş değeri: 5

`return s` : dönüş değeri **5** hazırlanır, ama metot hemen bitmez...

Adım 4 / 13

ÇAĞRI YIĞINI

`bol(10, 2)`

`main()`

ÇIKTI

```
try başladı
try bitti
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
▶      System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

bekleyen dönüş değeri: 5

…önce `finally` çalışır. catch atlandı (istisna yok).

Adım 5 / 13

ÇAĞRI YIĞINI

- `bol(10, 2)`
- `main()`

ÇIKTI

```
try başladı
try bitti
finally
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
▶    System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

bekleyen dönüş değeri: 5

Şimdi 5 döner ve yazdırılır.

Adım 6 / 13

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
▶    System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

dönüş değeri henüz yok

İkinci çağrı: `bol(10, 0)` .

Adım 7 / 13

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
▶      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)`

`finally { … }`

dönüş değeri henüz yok

try başlar.

Adım 8 / 13

ÇAĞRI YIĞINI

`bol(10, 0)`

`main()`

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
try başladı
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
▶      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

try { … }

catch (ArithmeticException e)

finally { … }

dönüş değeri henüz yok

`10 / 0` → istisna! try'ın kalanı (“try bitti”, `return s`) atlanır.

Adım 9 / 13

ÇAĞRI YIĞINI

- `bol(10, 0)`
- `main()`

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
try başladı
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
▶      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

try { … }

catch (ArithmeticException e) ✓

finally { … }

dönüş değeri henüz yok

catch eşleşir ve mesajı yazar.

Adım 10 / 13

ÇAĞRI YIĞINI

- bol(10, 0)
- main()

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
try başladı
catch: / by zero
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
▶      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)` ✓

`finally { … }`

bekleyen dönüş değeri: -1

`return -1` : değer hazır, ama yine önce finally.

Adım 11 / 13

ÇAĞRI YIĞINI

- `bol(10, 0)`
- `main()`

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
try başladı
catch: / by zero
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
▶      System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
16   System.out.println("Sonuç = " + bol(10, 0));
17 }
```

try { … }

catch (ArithmeticException e) ✓

finally { … }

bekleyen dönüş değeri: -1

`finally` bu kez de çalışır.

Adım 12 / 13

ÇAĞRI YIĞINI

- **bol(10, 0)**
- main()

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
try başladı
catch: / by zero
finally
```

finally ve return: Adım Adım

```
1  static int bol(int a, int b) {
2    try {
3      System.out.println("try başladı");
4      int s = a / b;
5      System.out.println("try bitti");
6      return s;
7    } catch (ArithmeticException e) {
8      System.out.println("catch: " + e.getMessage());
9      return -1;
10   } finally {
11     System.out.println("finally");
12   }
13 }
14 public static void main(String[] args) {
15   System.out.println("Sonuç = " + bol(10, 2));
▶    System.out.println("Sonuç = " + bol(10, 0));
17 }
```

`try { … }`

`catch (ArithmeticException e)` ✓

`finally { … }`

bekleyen dönüş değeri: -1

-1 döner. Sonuç: finally **iki durumda da** çalıştı.

Adım 13 / 13

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
try başladı
try bitti
finally
Sonuç = 5
try başladı
catch: / by zero
finally
Sonuç = -1
```

FINALLY

try-with-resources (Java 7+)

```
class Kaynak implements AutoCloseable {
  private String ad;
  Kaynak(String ad) {
    this.ad = ad;
    System.out.println(ad + " açıldı");
  }
  void kullan(int x) {
    System.out.println(ad + " kullanılıyor");
    int s = 10 / x;
  }
  @Override
  public void close() {
    System.out.println(ad + " kapatıldı");
  }
}
```

```
try (Kaynak k = new Kaynak("dosya")) {
  k.kullan(0);
} catch (ArithmeticException e) {
  System.out.println("Hata: " + e.getMessage());
}
```

Çıktı

```
dosya açıldı
dosya kullanılıyor
dosya kapatıldı
Hata: / by zero
```

- `try (...)` içinde açılan kaynak, blok bitince **otomatik** kapatılır (`close()`); istisna olsa bile.
- Kaynak **catch'ten önce** kapanır. Kaynak sınıfı `AutoCloseable` arayüzünü uygulamalıdır: akışlar, `Scanner` , NIO kanalları ve veri tabanı bağlantıları uygular; `finally` ve `null` denetimi gerekmez.
- Kaynak kapatmak için `finally` yazmanın modern ve güvenli yolu budur.

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
▶ try (Kaynak a = new Kaynak("dosya");
2       Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```

Kaynaklar (açılış sırası)



İlk kaynak oluşur: `dosya açıldı`.

Adım 1 / 8

ÇIKTI

```
dosya açıldı
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
1 try (Kaynak a = new Kaynak("dosya");
▶       Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```



İkinci kaynak oluşur. Kaynaklar **noktalı virgülle** ayrılır, yazılış sırasıyla açılır.

Adım 2 / 8

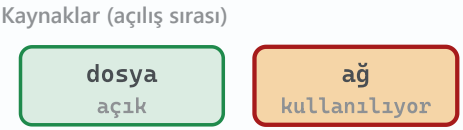
ÇIKTI

```
dosya açıldı
ağ açıldı
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
1 try (Kaynak a = new Kaynak("dosya");
2      Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```



`b.kullan(0)` : önce "ağ kullanılıyor" yazılır, sonra `10 / 0` .

Adım 3 / 8

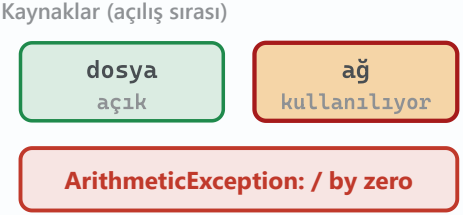
ÇIKTI

```
dosya açıldı
ağ açıldı
ağ kullanılıyor
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
1 try (Kaynak a = new Kaynak("dosya");
2      Kaynak b = new Kaynak("ağ")) {
▶   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```



İstisna fırlatıldı: try'ın kalanı ("try sonu") atlanır. Ama catch'ten **önce** kaynaklar kapatılacak.

Adım 4 / 8

ÇIKTI

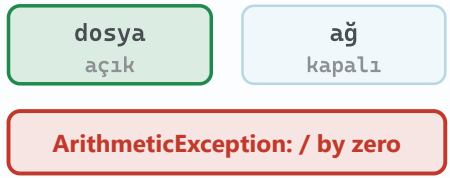
```
dosya açıldı
ağ açıldı
ağ kullanılıyor
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
1 try (Kaynak a = new Kaynak("dosya");
▶       Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```

Kaynaklar (açılış sırası)



Kapatma **ters sırayla**: son açılan `b` önce kapanır (`close()`).

Adım 5 / 8

ÇIKTI

```
dosya açıldı
ağ açıldı
ağ kullanılıyor
ağ kapatıldı
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
▶ try (Kaynak a = new Kaynak("dosya");
2       Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```

Kaynaklar (açılış sırası)

dosya	ağ
kapalı	kapalı

ArithmeticException: / by zero

Sonra `a` kapanır. Her iki kaynak da istisnaya rağmen kapandı.

Adım 6 / 8

ÇIKTI

```
dosya açıldı
ağ açıldı
ağ kullanılıyor
ağ kapatıldı
dosya kapatıldı
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
1 try (Kaynak a = new Kaynak("dosya");
2      Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
▶ } catch (ArithmeticException e) {
▶   System.out.println("Hata: " + e.getMessage());
7 } finally {
8   System.out.println("finally");
9 }
```

Kaynaklar (açılış sırası)

dosya	ağ
kapalı	kapalı

yakalandı

Şimdi catch çalışır: istisna yakalandı.

Adım 7 / 8

ÇIKTI

```
dosya açıldı
ağ açıldı
ağ kullanılıyor
ağ kapatıldı
dosya kapatıldı
Hata: / by zero
```

İki Kaynak: Açılış ve Kapanış Sırası

`Kaynak` : yapıcısı "... açıldı", `kullan` "... kullanılıyor", `close()` "... kapatıldı" yazar.

```
1 try (Kaynak a = new Kaynak("dosya");
2      Kaynak b = new Kaynak("ağ")) {
3   b.kullan(0);
4   System.out.println("try sonu");
5 } catch (ArithmeticException e) {
6   System.out.println("Hata: " + e.getMessage());
7 } finally {
▶   System.out.println("finally");
9 }
```

Kaynaklar (açılış sırası)

dosya	ağ
kapalı	kapalı

En son `finally` çalışır. Sıra: aç → kullan → kapat (ters) → catch → finally.

Adım 8 / 8

ÇIKTI

```
dosya açıldı
ağ açıldı
ağ kullanılıyor
ağ kapatıldı
dosya kapatıldı
Hata: / by zero
finally
```

İç İçe try Blokları

```
try {
  // Dıştaki try bloğu
  try {
    // İçteki try bloğu
    // Potansiyel hata içeren kodlar
  } catch (Exception e) {
    // İçteki catch bloğu
  } finally {
    // İçteki finally bloğu
  }
} catch (Exception e) {
  // Dıştaki catch bloğu
} finally {
  // Dıştaki finally bloğu
}
```

Bir `try` bloğu içinde başka bir `try` bloğu bulunabilir. İçteki try'da bir istisna oluşursa:

1. İçteki try'ın catch'lerinde **uygun** bir catch varsa o çalışır.
2. Yoksa istisna **dışa doğru** ilerler; ilk uygun catch bulunur ve çalışır.
3. Hiçbir catch eşleşmezse istisna metottan çıkar; sonunda kimse yakalamazsa **JVM'in varsayılan işleyicisi** programı sonlandırır.
4. Her durumda, istisnanın geçtiği her düzeydeki `finally` (varsa) çalışır.

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
▶      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```

- `dış try {`
  - `iç try { … s.length() }`
  - `iç catch (ArithmeticException)`
  - `iç finally`
  - `println("Dış try devam")`
- `dış catch (NullPointerException)`
- `dış finally`
- `println("Program sürüyor")`

Dış try başlar.

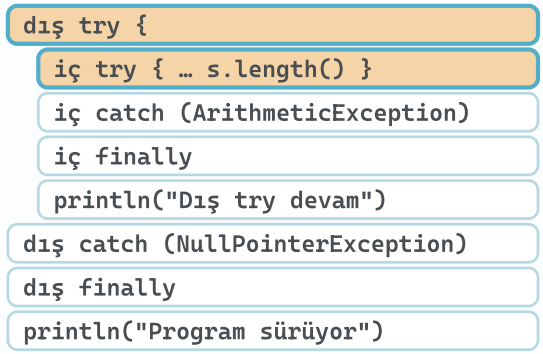
Adım 1 / 8

ÇIKTI

```
Dış try
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
▶        System.out.println("İç try");
▶        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```



İç try başlar; `s = null`.

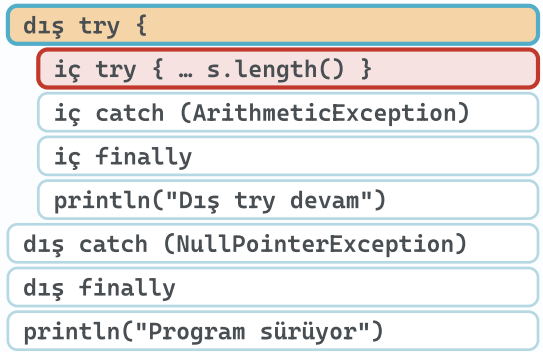
Adım 2 / 8

ÇIKTI

```
Dış try
İç try
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
▶        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```



`null` üzerinde metot çağrısı → `NullPointerException`.

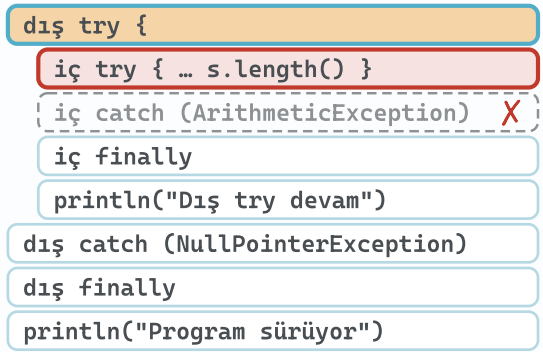
Adım 3 / 8

ÇIKTI

```
Dış try
İç try
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```



İç catch yalnızca `ArithmeticException` yakalar: **eşleşmez**.

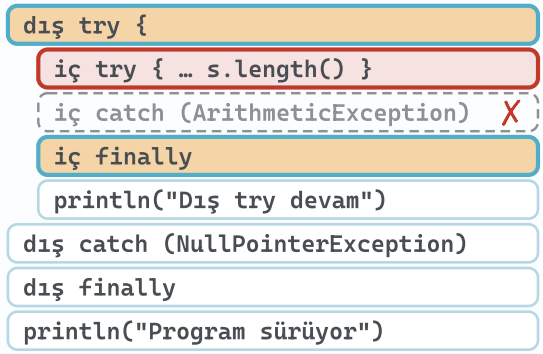
Adım 4 / 8

ÇIKTI

```
Dış try
İç try
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
▶        System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```



İstisna dışarı çıkmadan önce **iç finally** çalışır.

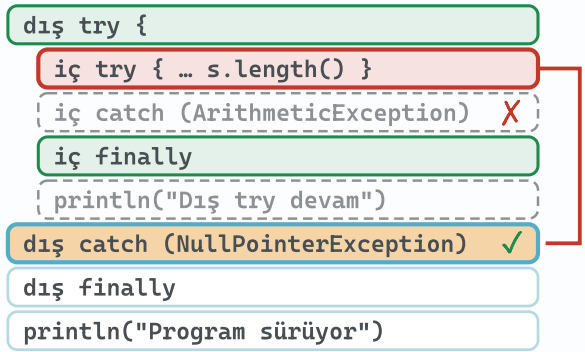
Adım 5 / 8

ÇIKTI

```
Dış try
İç try
İç finally
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
▶    } catch (NullPointerException e) {
▶      System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```



İstisna dış düzeye geçti; "Dış try devam" **atlandı**. Dış catch eşleşir.

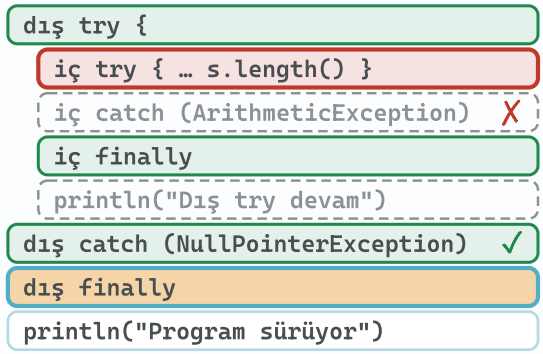
Adım 6 / 8

ÇIKTI

```
Dış try
İç try
İç finally
Dış catch: NPE
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
▶      System.out.println("Dış finally");
18   }
19   System.out.println("Program sürüyor");
20 }
```



Dış finally çalışır.

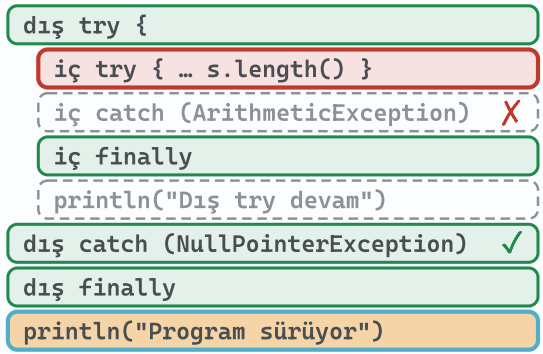
Adım 7 / 8

ÇIKTI

```
Dış try
İç try
İç finally
Dış catch: NPE
Dış finally
```

İç İçe try: İstisna Dışa Doğru İlerler

```
1  public static void main(String[] args) {
2    try {
3      System.out.println("Dış try");
4      try {
5        System.out.println("İç try");
6        String s = null;
7        System.out.println(s.length());
8      } catch (ArithmeticException e) {
9        System.out.println("İç catch");
10     } finally {
11       System.out.println("İç finally");
12     }
13     System.out.println("Dış try devam");
14   } catch (NullPointerException e) {
15     System.out.println("Dış catch: NPE");
16   } finally {
17     System.out.println("Dış finally");
18   }
▶  System.out.println("Program sürüyor");
20 }
```



İstisna ele alındı; program normal devam eder.

Adım 8 / 8

ÇIKTI

```
Dış try
İç try
İç finally
Dış catch: NPE
Dış finally
Program sürüyor
```

İstisnalar Hiyerarşisi

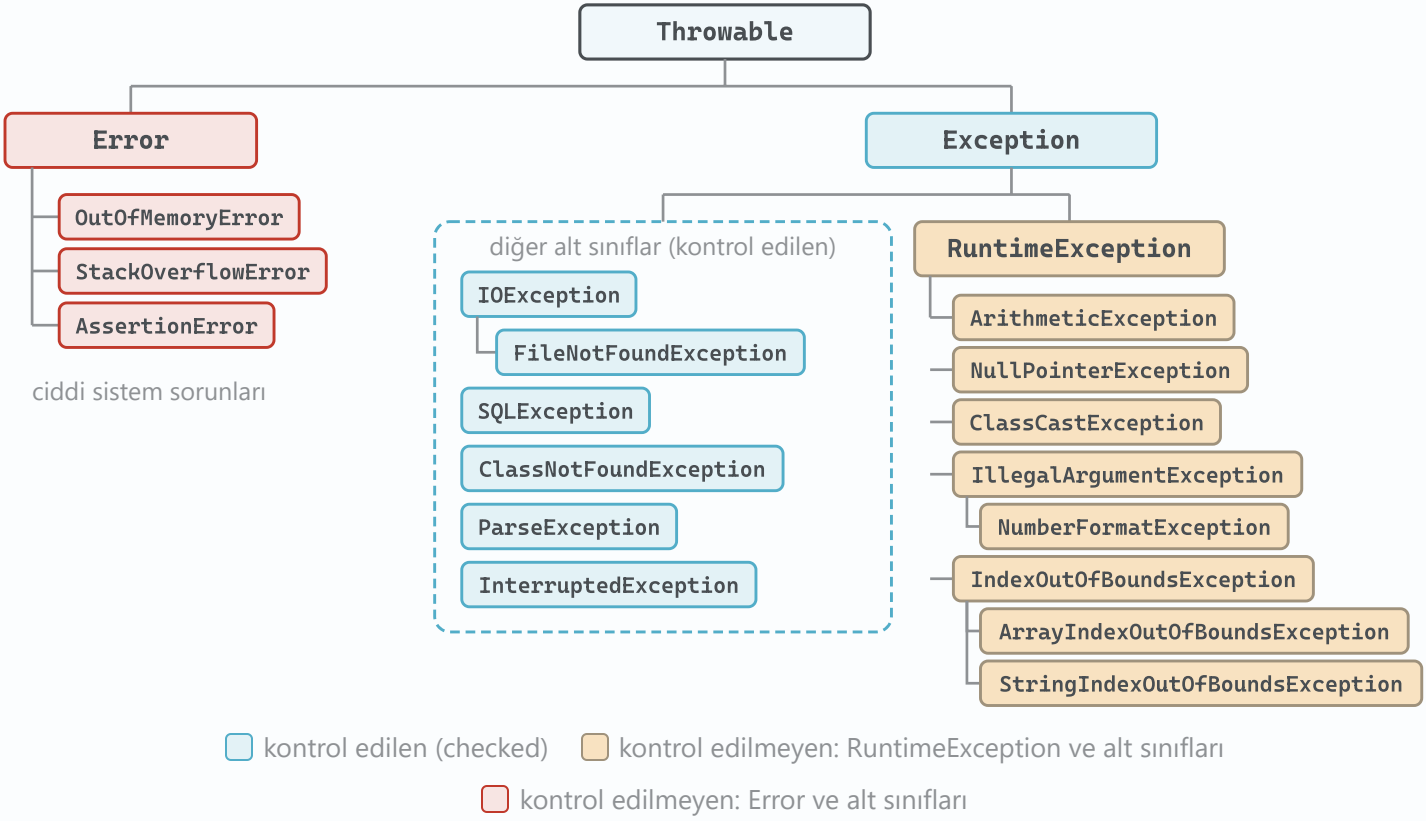
- Java'da istisna sınıfları bir **sınıf hiyerarşisi** içindedir.
- `Throwable` sınıfı, fırlatılabilen her şeyin **atasıdır**: yalnızca `Throwable` ve alt sınıflarının nesneleri `throw` edilebilir ve `catch` ile yakalanabilir.
- `Throwable`'ın iki doğrudan alt sınıfı vardır: `Error` ve `Exception`.
- Bir `catch` bloğu, yazılan tipi **ve tüm alt sınıflarını** yakalar (kalıtımın doğal sonucu).
- Java, yeni **özel istisna sınıfları** oluşturmayı destekler: uygulamaya özgü durumlara uyan sınıflar bu hiyerarşiye eklenir.

Neden hiyerarşi?

Aynı aileden istisnalar **tek bir üst tiple** yakalanabilir: `catch (IOException e)` hem `FileNotFoundException`'ı hem de diğer giriş/çıkış hatalarını yakalar.

İSTİSNA HİYERARŞİSİ

İstisna Sınıf Ağacı



İSTİSNA HİYERARŞİSİ

Error ve Exception

Error

- Programın **kurtarılamayacak** ciddi bir sorunla karşılaştığını belirtir.
- Bu durumlarla başa çıkmak çoğunlukla mümkün değildir; genellikle programın sonlanmasına yol açar.
- Örnek: program belleğinin tükenmesi (`OutOfMemoryError`), çağrı yığınının taşması, disk sürücüsünün "kaybolması" gibi aşırı durumlar.
- Uygulama kodunda **yakalanması önerilmez**.

Exception

+ Programın **öngörebileceği** ve çoğu zaman **kurtarabileceği** durumlar.
+ Dosya yok, ağ koptu, geçersiz girdi, `null` referans...
+ `try` - `catch` ile ele alınması beklenen tiptir.
+ Alt sınıfı `RuntimeException` , çoğunlukla **programcı hatalarını** temsil eder.

Her ikisi de `Throwable` 'dan türer; bu yüzden ikisi de fırlatılabilir. Ama "ele alınmalı" olan, `Exception` ailesidir.

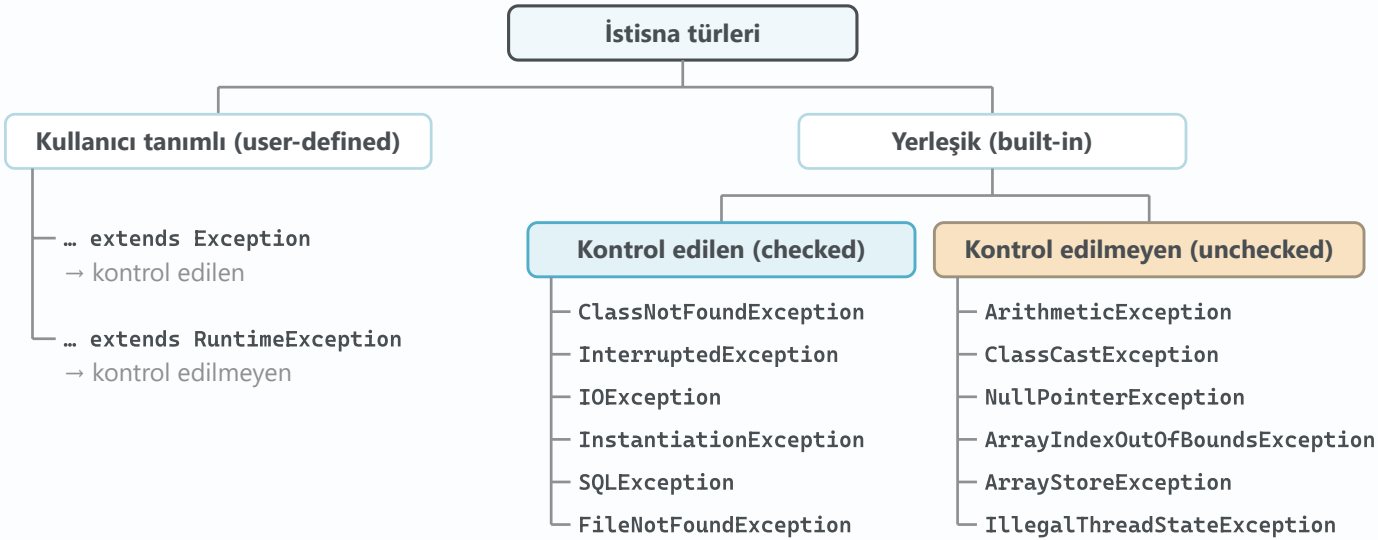
İstisna Türleri

Kontrol edilen (checked exceptions)

Derleme zamanında **derleyici denetler**; programcı ele almak ya da bildirmek zorundadır. Ortaya çıkabileceği önceden bilinen durumlar.

Kontrol edilmeyen (unchecked exceptions)

Derleme zamanında denetlenmez; çalışma zamanında ortaya çıkar. `RuntimeException`, `Error` ve alt sınıfları.



İSTİSNA TÜRLERİ

Kontrol Edilen İstisna (Checked)

- Kod tarafından **ele alınması veya bildirilmesi** gerekir: "catch or declare" (yakala ya da bildir) kuralı.
- Metodun başında `throws` ile belirtilmeli veya `try` - `catch` içinde ele alınmalıdır.
- `Exception` 'dan türer ama `RuntimeException` 'dan **türemez**.
- Bu kural **derleme zamanında** denetlenir: uyulmazsa program derlenmez.
- Örnek: var olmayan bir dosyayı okumaya çalışmak. Program ne kadar doğru yazılsa da dosya silinmiş olabilir; bu yüzden derleyici bir plan ister.

✗ NE ELE ALINMIŞ NE BİLDİRİLMİŞ

```
public void dosyaOku() {
  FileReader okuyucu = new FileReader("veri.txt");
}
```

```
error: unreported exception FileNotFoundException;
  must be caught or declared to be thrown
```

✓ THROWS İLE BİLDİRİLMİŞ

```
public void dosyaOku() throws FileNotFoundException {
  // Dosya okuma kodu
  FileReader okuyucu = new FileReader("veri.txt");
}
```

Kontrol Edilen İstisnayı Ele Almak

```java
public static void dosyaOku2() {
  try {
    FileReader okuyucu = new FileReader("veri.txt");
    // Dosya okuma kodu
  } catch (FileNotFoundException e) {
    // Kontrol edilen istisna burada ele alınıyor
    System.err.println("Dosya bulunamadı: " + e.getMessage());
    // Gerekirse istisna tekrar fırlatılabilir:
    // throw e;
  }
}
```

Çıktı (Windows; Linux'ta: No such file or directory)

```
Dosya bulunamadı: veri.txt (The system cannot find the file specified)
```

- Bu sürümde metot istisnayı **kendisi** yakalar; imzasında `throws` yoktur. Çağıran kod hiçbir şey yapmak zorunda değildir.
- `System.err` : hata çıktısı akışı. Konsolda çoğu IDE bunu kırmızı gösterir.
- Mesaj, işletim sisteminden gelir; Windows ve Linux'ta farklıdır.
- `throw e;` : yakalanan istisna, gerekirse (örneğin kayıt tuttuktan sonra) **tekrar fırlatılabilir**. O zaman metot yine `throws FileNotFoundException` bildirmelidir.

Kontrol Edilmeyen İstisna (Unchecked)

- Ele alınması veya bildirilmesi **zorunlu olmayan** istisna türleridir.
- Genellikle **mantık hataları** veya **programcı hataları** sonucu ortaya çıkar: sıfıra bölme, `null` kullanımı, dizi sınırını aşma.
- `throws` ile belirtilmesi veya `try-catch` içine alınması zorunlu değildir; derleyici sessiz kalır.
- `RuntimeException` veya `Error` sınıflarından türer.
- Doğru çözüm çoğu zaman yakalamak değil, **önlemektir**: bölmeden önce paydayı, kullanmadan önce `null`'ı denetlemek.

```
public void bolmeIslemi(int pay, int payda) {
  // Kontrol edilmeyen istisna: ArithmeticException
  int sonuc = pay / payda;
}
```

Derlenir, `throws` gerekmez. `payda == 0` olursa çalışma zamanında istisna oluşur.

```
public void bolmeIslemi(int pay, int payda) {
  if (payda == 0) {   // önleme
    System.out.println("Payda sıfır olamaz");
    return;
  }
  int sonuc = pay / payda;
}
```

İSTİSNA TÜRLERİ

Kontrol Edilen, Kontrol Edilmeyen, Error

	Kontrol edilen	Kontrol edilmeyen	Error
Üst sınıf	`Exception` (RuntimeException hariç)	`RuntimeException`	`Error`
Derleyici denetler mi?	**Evet**: ele al veya bildir	**Hayır**	**Hayır**
Tipik neden	Programın dışındaki koşullar (dosya, ağ, veri tabanı)	Programcı hatası (null, sınır, bölme)	JVM / sistem sorunu
Ne yapılmalı?	Yakala ve kurtar, ya da `throws` ile ilet	Kodu düzelt, girdiyi denetle	Genellikle yakalama; programı düzgün sonlandır
Örnekler	`IOException`, `SQLException`, `ClassNotFoundException`	`NullPointerException`, `ArithmeticException`	`OutOfMemoryError`, `StackOverflowError`

İstisna Fırlatma (Throwing an Exception)

- Java çalışma zamanı beklenmeyen bir durumla karşılaştığında, üst düzeydeki kodlara bildirmek için bir **istisna nesnesi oluşturur** ve **fırlatır**.
- Programcı da aynısını `throw` anahtar sözcüğüyle yapabilir.
- İstisna, normal program akışını **değiştirir**: `throw` 'dan sonraki satırlar çalışmaz; denetim uygun `catch` 'e geçer.
- Fırlatılabilen birçok tür vardır: `IOException` , `NullPointerException` , `IllegalArgumentException` , programcının kendi özel sınıfları...
- Fırlatılan istisna, çağıran kod tarafından `try` - `catch` ile yakalanabilir. Hiçbir yerde ele alınmazsa JVM programı sonlandırır ve **neyin** yanlış gittiğini, **nerede** olduğunu bildirir.

SÖZ DİZİMİ

```
throw new Exception("Bu bir istisna durumu!");
```

`new` ile istisna **nesnesi** oluşturulur (yapıcıya mesaj verilir), `throw` ile fırlatılır.

```
try {
  throw new Exception("Bu bir istisna durumu!");
} catch (Exception e) {
  System.out.println(e);
}
```

Çıktı

```
java.lang.Exception: Bu bir istisna durumu!
```

Durumu En İyi Anlatan İstisna Tipi

```
class CustomException extends Exception {
  public CustomException(String mesaj) {
    super(mesaj);
  }
}
public static void kontrolEt(int deger)
    throws CustomException {
  if (deger < 0) {
    // Belirli bir durumu kontrol et ve istisna fırlat
    throw new CustomException("Değer negatif olamaz!");
  }
  // Değer pozitifse, normal işlem devam eder
  System.out.println("Değer uygun: " + deger);
}
// main içinde
try {
  kontrolEt(5);
  kontrolEt(-3);
  kontrolEt(8);
} catch (CustomException e) {
  System.out.println("Yakalandı: " + e.getMessage());
}
```

Çıktı

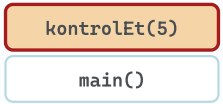
```
Değer uygun: 5
Yakalandı: Değer negatif olamaz!
```

- İstisna tipi seçilirken durumu **en iyi temsil eden** sınıf seçilmelidir.
- Özel senaryolar için **özel istisna sınıfları** oluşturulabilir.
- `CustomException` bir `Exception` alt sınıfıdır: **kontrol edilen** bir istisnadır. Bu yüzden metot imzasında `throws CustomException` şarttır.
- `kontrolEt(-3)` fırlattığı anda try'ın kalanı atlanır: `kontrolEt(8)` **hiç çağrılmaz**.

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
▶    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını



main, try içinde `kontrolEt(5)` 'i çağırır; yığına yeni çerçeve eklenir.

Adım 1 / 9

ÇAĞRI YIĞINI

`main()`

ÇIKTI

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
▶    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını

kontrolEt(5)

main()

`5 < 0` yanlış: throw satırı atlanır.

Adım 2 / 9

ÇAĞRI YIĞINI

kontrolEt(5)

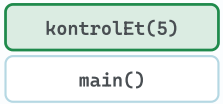
main()

ÇIKTI

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
▶    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını



“Değer uygun: 5” yazılır; metot normal biçimde döner.

Adım 3 / 9

ÇAĞRI YIĞINI

- kontrolEt(5)
- main()

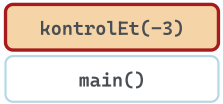
ÇIKTI

```
Değer
uygun: 5
```

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
▶    kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını



Sıradaki çağrı: `kontrolEt(-3)`.

Adım 4 / 9

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
Değer uygun:
5
```

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
▶    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını

kontrolEt(−3)

main()

`−3 < 0` doğru.

Adım 5 / 9

ÇAĞRI YIĞINI

kontrolEt(−3)

main()

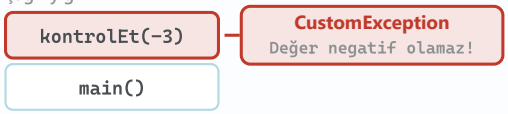
ÇIKTI

```
Değer
uygun: 5
```

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
▶      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını



`new` ile istisna nesnesi oluşur, `throw` fırlatır. 5. satır çalışmaz; metotta catch yok, çerçeve atılır.

Adım 6 / 9

ÇAĞRI YIĞINI

- `kontrolEt(-3)`
- `main()`

ÇIKTI

```
Değer uygun: 5
```

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
▶ } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını

`main()`

İstisna main'e geçti; try'ın kalanı (`kontrolEt(8)`) **atlanır**, uyan catch bulunur.

Adım 7 / 9

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
Değer uygun:
5
```

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
▶    System.out.println("Yakalandı: " + e.getMessage());
14 }
```

Çağrı yığını

`main()`

`getMessage()`, yapıcıya verilen mesajı döndürür.

Adım 8 / 9

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
Değer uygun: 5
Yakalandı: Değer negatif olamaz!
```

throw: Fırlatılan İstisna try'ın Kalanını Atlatır

```
1  static void kontrolEt(int deger) throws CustomException {
2    if (deger < 0) {
3      throw new CustomException("Değer negatif olamaz!");
4    }
5    System.out.println("Değer uygun: " + deger);
6  }
7  // main içinde
8  try {
9    kontrolEt(5);
10   kontrolEt(-3);
11   kontrolEt(8);
12 } catch (CustomException e) {
13   System.out.println("Yakalandı: " + e.getMessage());
▶ }
```

Çağrı yığını

`main()`

catch bitti; program normal akışına devam eder. `kontrolEt(8)` hiç çağrılmadı.

Adım 9 / 9

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
Değer uygun: 5
Yakalandı: Değer negatif olamaz!
```

İSTİSNA FIRLATMA

throw ve throws Farkı

	`throw`	`throws`
Görevi	Bir istisnayı **fırlatır** (şimdi)	Metodun fırlatabileceği istisnaları **bildirir** (ileride olabilir)
Yeri	Metot gövdesinde, bir deyim olarak	Metot imzasında, parametrelerden sonra
Ardından	Tek bir nesne: `throw new X("…");`	Bir veya daha çok tip: `throws X, Y`
Örnek	`throw new ArithmeticException("Sıfıra bölme");`	`void bol(int x, int y) throws ArithmeticException`

✗ THROWS GÖVDEDE KULLANILMAZ

```
if (x < 0) {
  throws new IllegalArgumentException("negatif");
}
```

```
error: illegal start of expression
```

✗ KONTROL EDİLEN İSTİSNA, BİLDİRİLMEDEN FIRLATILMIŞ

```
static void f(int x) {
  if (x < 0) throw new Exception("negatif");
}
```

```
error: unreported exception Exception;
  must be caught or declared to be thrown
```

İstisna Fırlatan Metotları Çağırmak

DOSYA OKUMA METODU

```
public void dosyaOku() throws FileNotFoundException {
  // Dosya okuma kodu
}
```

1) ÇAĞIRAN ELE ALIR (TERCİH EDİLEN)

```
try {
  dosyaOku();
} catch (FileNotFoundException e) {
  // Dosya bulunamadı durumuyla başa çıkma kodu
  System.err.println("Dosya bulunamadı: " + e.getMessage());
}
```

2) ÇAĞIRAN DA BİLDİRİR (İLETİR)

```
public void raporHazirla() throws FileNotFoundException {
  dosyaOku();
}
```

- Metot çağırırken, metodun **imzasında belirtilen** istisnalar dikkate alınmalı ve uygun bir strateji seçilmelidir.
- Çağrı, uygun `try` - `catch` bloğunun **içine** yazılır.
- Ya da çağıran metot da imzasında `throws` ile aynı tipi bildirir; sorun bir üst düzeye geçer. Bu, sorunu yalnızca ertelediği için yakalamaktan **daha az tercih edilir**.
- `main` bile `throws` yazabilir; o zaman yakalanmayan istisna programı sonlandırır.

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
▶      int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)

`main()`

main, try içinde `hesapla(10)` 'u çağırır.

Adım 1 / 9

ÇAĞRI YIĞINI

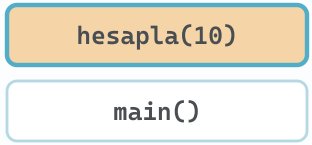
`main()`

ÇIKTI

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
▶    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)



Yığına `hesapla` çerçevesi eklendi.

Adım 2 / 9

ÇAĞRI YIĞINI

- `hesapla(10)`
- `main()`

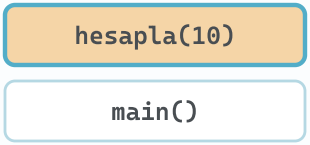
ÇIKTI

```
hesapla başladı
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
▶    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)



`hesapla` , `bol(10, 0)` 'ı çağırır.

Adım 3 / 9

ÇAĞRI YIĞINI

- `hesapla(10)`
- `main()`

ÇIKTI

```
hesapla başladı
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
▶    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)

- `bol(10, 0)`
- `hesapla(10)`
- `main()`

Yığının tepesinde artık `bol` var.

Adım 4 / 9

ÇAĞRI YIĞINI

- `bol(10, 0)`
- `hesapla(10)`
- `main()`

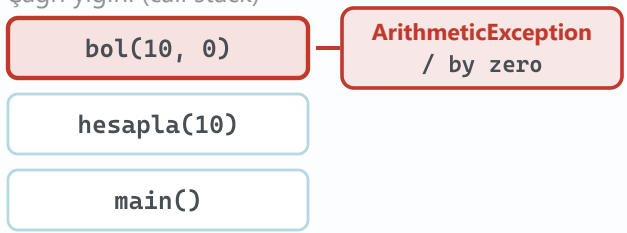
ÇIKTI

```
hesapla başladı
bol başladı
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
▶    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)



`10 / 0` : istisna **bol**'da doğdu. bol'da try yok → bol'un kalanı çalışmaz, çerçevesi **atılır**.

Adım 5 / 9

ÇAĞRI YIĞINI

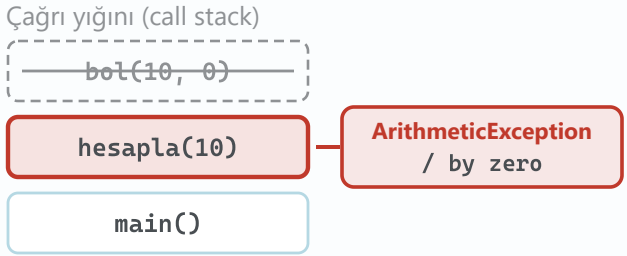
- `bol(10, 0)`
- `hesapla(10)`
- `main()`

ÇIKTI

```
hesapla başladı
bol başladı
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
▶    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```



İstisna çağıran yere, `hesapla`'ya geçti. Burada da try yok → "hesapla bitti" atlanır, çerçeve atılır.

Adım 6 / 9

ÇAĞRI YIĞINI

- **hesapla(10)**
- main()

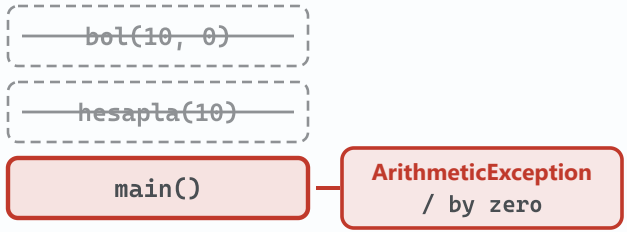
ÇIKTI

```
hesapla başladı
bol başladı
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
▶      int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)



İstisna `main`'e ulaştı. Çağrı bir `try` içinde: uygun catch aranır. "Sonuç" satırı atlanır.

Adım 7 / 9

ÇAĞRI YIĞINI

`main()`

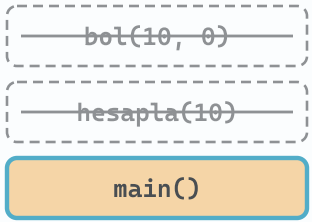
ÇIKTI

```
hesapla başladı
bol başladı
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
▶    } catch (ArithmeticException e) {
▶      System.out.println("main: " +
e.getMessage());
19   }
20   System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)



catch eşleşti: istisna **yakalandı**, yayılım durdu.

Adım 8 / 9

ÇAĞRI YIĞINI

`main()`

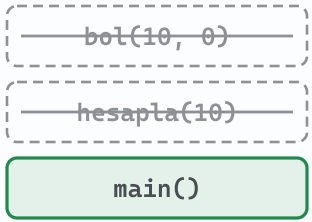
ÇIKTI

```
hesapla başladı
bol başladı
main: / by zero
```

Yayılım: İstisna Çağrı Yığınında İner

```
1  static int bol(int a, int b) {
2    System.out.println("bol başladı");
3    int s = a / b;
4    System.out.println("bol bitti");
5    return s;
6  }
7  static int hesapla(int x) {
8    System.out.println("hesapla başladı");
9    int r = bol(x, 0);
10   System.out.println("hesapla bitti");
11   return r;
12 }
13 public static void main(String[] args) {
14   try {
15     int sonuc = hesapla(10);
16     System.out.println("Sonuç: " + sonuc);
17   } catch (ArithmeticException e) {
18     System.out.println("main: " +
   e.getMessage());
19   }
▶    System.out.println("Program bitti");
21 }
```

Çağrı yığını (call stack)



Program normal biçimde biter. Kimse yakalamasaydı JVM programı sonlandırırdı.

Adım 9 / 9

ÇAĞRI YIĞINI

`main()`

ÇIKTI

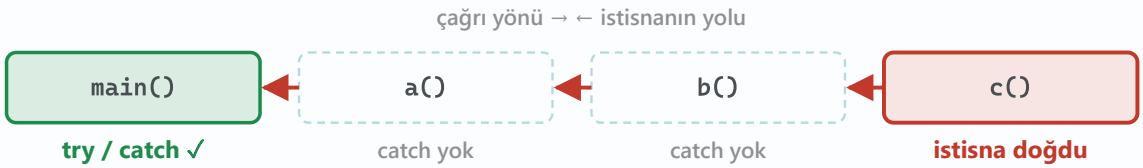
```
hesapla başladı
bol başladı
main: / by zero
Program bitti
```

İSTİSNA FIRLATMA

JVM İstisnayı Nasıl Ele Alır?

1. JVM (veya `throw`) bir **istisna nesnesi** oluşturur: tipi, mesajı ve oluştuğu yerin yığın izi (stack trace) içindedir.
2. İstisnanın oluştuğu metotta, onu saran bir `try` ve **uyumlu** bir `catch` aranır.
3. Bulunamazsa metot o anda sona erer (varsa `finally` çalışır) ve istisna **çağıran metoda** geçer. Arama, çağrı yığınında aşağıya doğru sürer.
4. Uygun bir catch bulunursa o çalışır; program catch'ten sonra devam eder.
5. `main` de yakalamazsa JVM'in **varsayılan işleyicisi** yığın izini yazdırır ve programı (o iş parçacığını) sonlandırır.

Bu arama sürecine **yığın çözülmesi** (stack unwinding) denir. Yakalanan istisna nesnesi catch bloğu bitince artık kullanılmaz ve çöp toplayıcı (garbage collector) tarafından temizlenir.



RuntimeException Sınıfları

`RuntimeException` ve alt sınıfları, **programcı hatası** veya **mantık hatası** sonucu oluşan istisnaları temsil eder. Derleme zamanında denetlenmez, ele alınmaları zorunlu değildir; ama programcının bu durumları **bilinçli olarak** önlemesi ya da ele alması gerekir.

Sınıf	Ne zaman oluşur?	Örnek
`NullPointerException`	`null` bir referansın üyesi kullanılınca	`String s = null; s.length();`
`ArrayIndexOutOfBoundsException`	Dizi sınırları dışında indis	`int[] d = new int[3]; d[5] = 1;`
`ArithmeticException`	Tam sayı sıfıra bölme, kesin aritmetikte taşma	`10 / 0`
`IllegalArgumentException`	Metoda geçersiz argüman verilince	`new ArrayList<>(-1)`
`NumberFormatException`	Metin geçerli bir sayı değilken dönüştürme	`Integer.parseInt("abc")`
`ClassCastException`	Nesne, dönüştürülmek istenen tipte değilken	`(Integer) nesne`

NullPointerException

- `null` bir referans üzerinden metot çağırmak, alana erişmek veya `length` almak bu istisnaya yol açar.
- En sık hata kaynaklarındandır. Önlemek için:
  - `if (str != null)` ile açık denetim
  - `Objects.requireNonNull(x, mesaj)` : `null` ise **hemen**, anlamlı mesajla NPE fırlatır
  - `Objects.requireNonNullElse(x, varsayilan)` (Java 9+): `null` ise varsayılan değeri döndürür
  - `Optional` sınıfı (Java 8+): "değer olmayabilir" durumunu açıkça temsil eder
  - Kütüphaneler: Apache Commons Lang `StringUtils` , `null` -güvenli metin işlemleri sunar

```
String str = null;
int length = str.length();
```

Çıktı (Java 14+ açıklayıcı mesaj)

```
Exception in thread "main"
java.lang.NullPointerException:
  Cannot invoke "String.length()" because
"str" is null
```

Java 14+ mesajı **hangi** referansın `null` olduğunu söyler. Değişken adı hata ayıklama bilgisiyle derlenince (`javac -g` , IDE'ler) görünür; yoksa `"<local1>"` .

null ile Güvenli Çalışmak

```
1  String str = null;
2  int length = str.length();
3  Objects.requireNonNull(str, "Str null olamaz");
4  Optional<String> optionalStr = Optional.ofNullable(str);
5  length = optionalStr.map(String::length).orElse(0);
6  String defaultValue = Objects.requireNonNullElse(str, "0");
7  if (str != null) {
8    System.out.println(str.length());
9  }
10 System.out.println("Ali".equals(str));
```

Satır	Sonuç (her satır tek başına denendiğinde)
2	**NullPointerException**: Cannot invoke "String.length()" because "str" is null
3	**NullPointerException**: Str null olamaz
5	**length = 0** (değer yok → `orElse(0)`)
6	**defaultValue = "0"**
7–9	Koşul yanlış; hiçbir şey yazılmaz, istisna yok
10	**false**: sabit metin önde, `null` argüman sorun değil

`"Ali".equals(str)` güvenlidir; `str.equals("Ali")` ise `str` `null` olduğunda NPE fırlatır. `requireNonNull`, hatayı **kaynağında** ve anlaşılır bir mesajla yakalamak için metot başlarında kullanılır.

ArrayIndexOutOfBoundsException

- Dizinin sınırları dışındaki bir indise erişme durumudur: geçerli indisler `0` … `length - 1`; **negatif** indis de geçersizdir.
- `IndexOutOfBoundsException` ailesinin bir üyesidir. Sınır aşımında aynı aileden farklı sınıflar fırlatılabilir:
  - dizi (`int[]`), `Vector`, `Arrays.asList(...)` listesi → `ArrayIndexOutOfBoundsException`
  - `ArrayList.get` → `IndexOutOfBoundsException` (üst sınıf)
  - `String.charAt` → `StringIndexOutOfBoundsException`
- Hepsini birden yakalamak için: `catch (IndexOutOfBoundsException e)`.

ÖNLEME: DÖNGÜ SINIRI `length` İLE YAZILIR

```
int[] numbers = {1, 2, 3};
for (int i = 0; i < numbers.length; i++) {
  System.out.print(numbers[i] + " ");
}
```

Çıktı

```
1 2 3
```

En sık neden: `i <= dizi.length` yazmak. Son tur `dizi[length]`'e erişir ve istisna oluşur.

Sınır Aşımı: Hangi Satır Hangi İstisna?

```
1  int[] numbers = {1, 2, 3};
2  int value = numbers[5];
3  ArrayList<String> list = new ArrayList<>();
4  String svalue = list.get(5);
5  Vector<Integer> vector = new Vector<>();
6  value = vector.get(10);
7  String[] names = {"Alice", "Bob", "Charlie"};
8  svalue = Arrays.asList(names).get(5);
9  String str = "Hello";
10 char ch = str.charAt(10);
```

Satır	Fırlatılan istisna ve mesajı
2	`ArrayIndexOutOfBoundsException` Index 5 out of bounds for length 3
4	`IndexOutOfBoundsException` Index 5 out of bounds for length 0
6	`ArrayIndexOutOfBoundsException` Array index out of range: 10
8	`ArrayIndexOutOfBoundsException` Index 5 out of bounds for length 3
10	`StringIndexOutOfBoundsException` Index 10 out of bounds for length 5

Her satır tek başına denendiğinde. Boş bir `ArrayList` 'in kapasitesi olsa bile **boyutu** 0'dır; `get` boyuta göre denetler.

ArithmeticException

- Matematiksel işlemler sırasında oluşan bir hatayı temsil eder; en bilinen örneği **tam sayıyı sıfıra bölmek** (`/` ve `%`).
- `Math` sınıfının "kesin" (exact) metotları taşmada fırlatır: `addExact` , `multiplyExact` , `toIntExact` ...
- `BigDecimal` yüksek hassasiyetli aritmetik sağlar; sıfıra bölmede veya sonucu tam gösterilemeyen bölmede (1/3) fırlatır.
- `double` işlemleri istisna fırlatmaz: `10.0 / 0` → `Infinity` , `Math.sqrt(-1)` → `NaN` .
- `Random.nextInt(0)` ise `ArithmeticException` değil, `IllegalArgumentException` fırlatır: sınır pozitif olmalıdır.

Sessiz taşma: sıradan `int` toplaması taşınca istisna **oluşmaz**, değer başa sarar: `Integer.MAX_VALUE + 1` → `-2147483648` . Taşmanın fark edilmesi gerekiyorsa `Math.addExact` kullanılır.

Aritmetik Hatalar: Satır Satır

```
1  int result = 10 / 0;
2  double dresult = Math.sqrt(-1);
3  System.out.println(dresult);
4  System.out.println(10.0 / 0);
5  int t = Math.addExact(Integer.MAX_VALUE, 1);
6  Random random = new Random();
7  int randomNumber = random.nextInt(0);
8  BigDecimal num1 = new BigDecimal("10");
9  BigDecimal num2 = BigDecimal.ZERO;
10 BigDecimal bresult = num1.divide(num2);
```

Satır	Sonuç
1	`ArithmeticException` : / by zero
3	**NaN** yazar, istisna yok
4	**Infinity** yazar, istisna yok
5	`ArithmeticException` : integer overflow
7	`IllegalArgumentException` : bound must be positive
10	`ArithmeticException` : Division by zero

Her satır tek başına denendiğinde. Kayan noktalı sayılarda sıfıra bölme IEEE 754 kurallarına göre özel değerler (`Infinity` , `NaN`) üretir; bu yüzden `double` sonuçlarını ayrıca denetlemek gerekir (`Double.isNaN`).

IllegalArgumentException

- Bir metodun **geçersiz bir argüman** aldığını belirtir. Kontrol edilmeyen bir istisnadır.
- Argümanları denetleyip fırlatmak, hatayı **kaynağında** durdurur.
- Kütüphane örnekleri:
  - `Arrays.copyOfRange(d, 5, 2)` : başlangıç > bitiş
  - `new ArrayList<>(-1)` : negatif kapasite
  - `Thread` 'in `setPriority(11)` : öncelik 1–10 arası olmalı
  - `new File(uri)` : adres `file:` şeklinde değilse

```
public void yasAyarla(int age) {
  if (age < 0) {
    throw new IllegalArgumentException("Yaş negatif olamaz");
  }
  this.age = age;
}
public void adAyarla(String name) {
  if (name.length() < 3) {
    throw new IllegalArgumentException("3 karakter olmalı");
  }
  this.name = name;
}
```

Her geçersiz durum bu istisna değildir: `new FileReader("invalid/file/path")` → `FileNotFoundException` ; `new Thread(null)` hiç fırlatmaz. Alt sınıfı: `NumberFormatException` .

Geçersiz Argüman: Satır Satır

```
1 yasAyarla(-5);
2 adAyarla("Al");
3 int[] numbers = {1, 2, 3};
4 Arrays.copyOfRange(numbers, 5, 2);
5 List<Integer> liste = new ArrayList<>(-1);
6 File file = new File(URI.create("http://ornek.com/a.txt"));
7 Thread thread = new Thread(gorev);
8 thread.setPriority(11);
```

Satır	Fırlatılan istisna (her satır tek başına)
1	IllegalArgumentException: Yaş negatif olamaz
2	IllegalArgumentException: 3 karakter olmalı
4	IllegalArgumentException: 5 > 2
5	IllegalArgumentException: Illegal Capacity: -1
6	IllegalArgumentException: URI scheme is not "file"
8	IllegalArgumentException (mesajsız)

İyi bir mesaj, geçersiz değeri ve beklenen aralığı söyler: `"Yaş negatif olamaz: " + age`. Mesajsız istisna (8. satır) hata ayıklamayı zorlaştırır.

NumberFormatException

- Bir metin sayıya dönüştürülürken metin **geçerli bir sayıyı temsil etmiyorsa** oluşur.
- `Integer.parseInt`, `Double.parseDouble`, `Float.parseFloat`, `Short.parseShort`, `Long.parseLong`, `Byte.parseByte` ve `valueOf` metotları fırlatabilir.
- Değer tipin aralığı dışındaysa da oluşur: `Byte.parseByte("300")`.
- `IllegalArgumentException`'ın alt sınıfıdır.
- Kullanıcı girdisi her zaman bu riski taşır: ya önceden denetlenir ya da yakalanır.

```
String strNumber = "abc";
int number = Integer.parseInt(strNumber);
strNumber = "123abc";
number = Integer.parseInt(strNumber);
String strDouble = "12.34abc";
double doubleValue = Double.parseDouble(strDouble);
String strShort = "45xyz";
short shortValue = Short.parseShort(strShort);
```

Çağrı	NumberFormatException mesajı
`Integer.parseInt("abc")`	For input string: "abc"
`Integer.parseInt("123abc")`	For input string: "123abc"
`Double.parseDouble("12.34abc")`	For input string: "12.34abc"
`Short.parseShort("45xyz")`	For input string: "45xyz"

Güvenli Sayı Dönüştürme

```
static int guvenliOku(String metin, int varsayilan) {
  try {
    return Integer.parseInt(metin.trim());
  } catch (NumberFormatException e) {
    return varsayilan;
  }
}
// main içinde
System.out.println(guvenliOku(" 42 ", 0));
System.out.println(guvenliOku("kırk", 0));
System.out.println(guvenliOku("10.5", -1));
```

Çıktı

```
42
0
-1
```

- Dönüştürme bir yardımcı metotta **tek bir yerde** denetlenir; geçersiz girdi için varsayılan değer döner.
- `trim()`, baştaki ve sondaki boşlukları siler: `parseInt(" 42 ")` tek başına istisna fırlatırdı.
- `"10.5"` bir `int` değildir; ondalık sayı için `Double.parseDouble` kullanılır.
- Varsayılan değer, geçerli bir sonuçla karışmayacak biçimde seçilmelidir; karışabiliyorsa istisnayı çağırana iletmek daha doğrudur.

parseInt ve catch: Üç Girdi

```
1  static int guvenliOku(String metin, int varsayilan) {
2    try {
3      return Integer.parseInt(metin.trim());
4    } catch (NumberFormatException e) {
5      return varsayilan;
6    }
7  }
▶ System.out.println(guvenliOku(" 7 ", 0));
9  System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin



İlk çağrı: metin " 7 " (başta ve sonda boşluk), varsayılan 0.

Adım 1 / 8

DEĞİŞKENLER

—

ÇIKTI

parseInt ve catch: Üç Girdi

```
1 static int guvenliOku(String metin, int varsayilan) {
2   try {
▶       return Integer.parseInt(metin.trim());
4   } catch (NumberFormatException e) {
5     return varsayilan;
6   }
7 }
8 System.out.println(guvenliOku(" 7 ", 0));
9 System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin

·	7	·

trim()

7

parseInt("7") = 7 → return 7

`trim()` boşlukları atar; "7" sayıya çevrilir, try içinden 7 döner.

Adım 2 / 8

DEĞİŞKENLER

`metin = " 7 "` `varsayilan = 0`

ÇIKTI

```
7
```

parseInt ve catch: Üç Girdi

```
1  static int guvenliOku(String metin, int varsayilan) {
2    try {
3      return Integer.parseInt(metin.trim());
4    } catch (NumberFormatException e) {
5      return varsayilan;
6    }
7  }
8  System.out.println(guvenliOku(" 7 ", 0));
▶  System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin

y	e	d	i

İkinci çağrı: "yedi".

Adım 3 / 8

DEĞİŞKENLER

—

ÇIKTI

```
7
```

parseInt ve catch: Üç Girdi

```
1 static int guvenliOku(String metin, int varsayilan) {
2   try {
▶       return Integer.parseInt(metin.trim());
4   } catch (NumberFormatException e) {
5     return varsayilan;
6   }
7 }
8 System.out.println(guvenliOku(" 7 ", 0));
9 System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin

y	e	d	i

trim()

y	e	d	i

NumberFormatException: For input string: "yedi"

`parseInt` rakam olmayan metinde istisna fırlatır; `return` çalışamadı.

Adım 4 / 8

DEĞİŞKENLER

`metin = "yedi"` `varsayilan = 0`

ÇIKTI

```
7
```

parseInt ve catch: Üç Girdi

```
1  static int guvenliOku(String metin, int varsayilan) {
2    try {
3      return Integer.parseInt(metin.trim());
▶    } catch (NumberFormatException e) {
▶      return varsayilan;
6    }
7  }
8  System.out.println(guvenliOku(" 7 ", 0));
9  System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin

y	e	d	i

trim()

y	e	d	i

catch → return varsayilan = 0

catch eşleşti: varsayılan değer 0 döner. İstisna metot dışına çıkmaz.

Adım 5 / 8

DEĞİŞKENLER

`metin = "yedi"` `varsayilan = 0`

`e = NumberFormatException`

ÇIKTI

```
7
0
```

parseInt ve catch: Üç Girdi

```
1 static int guvenliOku(String metin, int varsayilan) {
2   try {
3     return Integer.parseInt(metin.trim());
4   } catch (NumberFormatException e) {
5     return varsayilan;
6   }
7 }
8 System.out.println(guvenliOku(" 7 ", 0));
9 System.out.println(guvenliOku("yedi", 0));
▶ System.out.println(guvenliOku("7.5", -1));
```

metin

7	.	5

Üçüncü çağrı: "7.5", varsayılan −1.

Adım 6 / 8

DEĞİŞKENLER

—

ÇIKTI

```
7
0
```

parseInt ve catch: Üç Girdi

```
1  static int guvenliOku(String metin, int varsayilan) {
2    try {
▶      return Integer.parseInt(metin.trim());
4    } catch (NumberFormatException e) {
5      return varsayilan;
6    }
7  }
8  System.out.println(guvenliOku(" 7 ", 0));
9  System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin

7	.	5

trim()

7	.	5

NumberFormatException: For input string: "7.5"

Nokta bir rakam değildir: "7.5" geçerli bir `int` değil.

Adım 7 / 8

DEĞİŞKENLER

`metin = "7.5"` `varsayilan = -1`

ÇIKTI

```
7
0
```

parseInt ve catch: Üç Girdi

```
1  static int guvenliOku(String metin, int varsayilan) {
2    try {
3      return Integer.parseInt(metin.trim());
▶    } catch (NumberFormatException e) {
▶      return varsayilan;
6    }
7  }
8  System.out.println(guvenliOku(" 7 ", 0));
9  System.out.println(guvenliOku("yedi", 0));
10 System.out.println(guvenliOku("7.5", -1));
```

metin

7	.	5

trim()

7	.	5

catch → return varsayilan = -1

−1 döner. Çağıran, istisna yerine varsayılan değerle devam eder.

Adım 8 / 8

DEĞİŞKENLER

`metin = "7.5"` `varsayilan = -1`

`e = NumberFormatException`

ÇIKTI

```
7
0
-1
```

ClassCastException

- Bir nesne, dönüştürülmek istendiği tipten **değilken** tip dönüşümü (cast) yapılınca oluşur.
- Değişkenin tipi değil, **nesnenin gerçek tipi** belirleyicidir: `Object obj = "Merhaba"` bir `String` nesnesidir.
- Ham (raw) tipli koleksiyonlar (`ArrayList` , `Map` , `Set`) her türü kabul eder; yanlış tipte öğe **alınırken** dönüşüm başarısız olur. Sıralı koleksiyonlar (`TreeSet` , `TreeMap`) karşılaştırılamayan tipte öğe **eklenirken** de fırlatır.
- Önleme: **jenerikler** (`ArrayList<String>`) yanlış tipi **derleme zamanında** yakalar; dönüşümden önce `instanceof` ile denetlenir.

```
1  Object obj = "Merhaba";
2  Integer number = (Integer) obj;
3  ArrayList list = new ArrayList();
4  list.add("Merhaba");
5  number = (Integer) list.get(0);
6  Map map = new HashMap();
7  map.put("anahtar", "Değer");
8  Integer value = (Integer) map.get("anahtar");
9  Set set = new HashSet();
10 set.add("Merhaba");
11 value = (Integer) set.iterator().next();
12 Object[] objArray = new String[5];
13 Integer[] intArray = (Integer[]) objArray;
```

2, 5, 8, 11 ve 13. satırların her biri `ClassCastException` fırlatır, örneğin:

```
class java.lang.String cannot be cast to class
java.lang.Integer
```

Dönüşümden Önce instanceof

KLASİK BİÇİM

```
Object obj = "Merhaba";
if (obj instanceof Integer) {
  Integer sayi = (Integer) obj;
  System.out.println("Sayı: " + sayi);
} else {
  System.out.println("obj bir Integer değil");
}
```

Çıktı

```
obj bir Integer değil
```

DESEN EŞLEME (JAVA 16+)

```
if (obj instanceof Integer sayi) {
  System.out.println("Sayı: " + sayi);
}
```

JENERİKLER İLE DERLEME ZAMANINDA

```
ArrayList<String> liste = new ArrayList<>();
liste.add("Merhaba");
Integer n = (Integer) liste.get(0);
```

```
error: incompatible types: String cannot be
  converted to Integer
```

Çalışma zamanı istisnası yerine **derleme hatası**: hata daha erken, programı çalıştırmadan bulunur. Bu yüzden ham tipler yerine jenerik tipler tercih edilir.

KONTROL EDİLEN SINIFLAR

Kontrollü İstisna Sınıfları

Kodun hatayı ele alması ve uygun biçimde **kurtarması** gereken, çoğunlukla programın **dışındaki** koşullardan doğan durumları temsil eder. Derleyici ele alınmalarını şart koşar.

Sınıf	Paket	Ne zaman?
`FileNotFoundException`	`java.io`	Açılmak istenen dosya yok veya açılamıyor
`IOException`	`java.io`	Giriş/çıkış işlemlerinde genel hata (üst sınıf)
`SQLException`	`java.sql`	Veri tabanı erişiminde hata
`ParseException`	`java.text`	Metni bir biçime göre ayrıştırırken hata
`ClassNotFoundException`	`java.lang`	Adıyla yüklenmek istenen sınıf bulunamadı
`InterruptedException`	`java.lang`	Bekleyen iş parçacığı kesildi (`Thread.sleep`)
`NotSerializableException`	`java.io`	Serileştirilemeyen nesne yazılmak istendi

KONTROL EDİLEN SINIFLAR

FileNotFoundException

```
try {
  FileReader fileReader = new FileReader("dosya.txt");
} catch (FileNotFoundException e) {
  // Dosya bulunamadı durumuyla ilgili işlemler
  System.out.println(e);
}
```

Çıktı (dosya yokken, Windows)

```
java.io.FileNotFoundException: dosya.txt
  (The system cannot find the file specified)
```

- Bir dosya açılmak istendiğinde dosya **bulunamazsa** (ya da bir klasörse, okuma izni yoksa) oluşur.
- `IOException` 'ın alt sınıfıdır; `catch (IOException e)` onu da yakalar.
- Göreli yol (`"dosya.txt"`), programın **çalışma klasörüne** göre aranır. IDE'lerde bu genellikle proje klasörüdür: dosya "yanlış klasörde" olabilir.
- Tipik tepki: kullanıcıya bildirip yeni bir ad istemek veya varsayılan değerlerle devam etmek.

IOException

```
try {
  BufferedReader reader =
    new BufferedReader(new FileReader(path));
  String line;
  while ((line = reader.readLine()) != null) {
    System.out.println(line);
  }
  reader.close();
} catch (IOException e) {
  // IOException durumuyla ilgili işlemler
  e.printStackTrace();
}
```

Çıktı (notlar.txt varken)

```
birinci satır
ikinci satır
```

- Giriş/çıkış işlemlerinde oluşan **genel** hatayı temsil eder; `FileNotFoundException` gibi birçok alt sınıfı vardır.
- **Sorun**: `readLine()` istisna fırlatırsa `reader.close()` hiç çalışmaz; dosya açık kalır (kaynak sızıntısı).

✓ TRY-WITH-RESOURCES İLE

```
try (BufferedReader okuyucu =
    new BufferedReader(new FileReader(path))) {
  String satir;
  while ((satir = okuyucu.readLine()) != null) {
    System.out.println(satir);
  }
} catch (IOException e) {
  System.out.println(e.getMessage());
}
```

SQLException

```
try {
  Connection connection =
    DriverManager.getConnection("jdbc:mysql://localhost/okul");
  PreparedStatement preparedStatement =
    connection.prepareStatement("SELECT user_id FROM users");
  // Sorguyu çalıştırma ve sonuçları alıp işleme
  ResultSet resultSet = preparedStatement.executeQuery();
  while (resultSet.next()) {
    // Sonuçları işleme
    System.out.println("User ID: " + resultSet.getInt("user_id"));
  }
  // Bağlantıyı kapatma
  connection.close();
} catch (SQLException e) {
  // SQLException durumuyla ilgili işlemler
  System.out.println(e);
}
```

- Veri tabanı işlemleri sırasında bir hata oluştuğunda fırlatılır: bağlantı kurulamadı, SQL söz dizimi yanlış, tablo yok, sürücü yok...
- `getMessage()`, `getSQLState()` ve `getErrorCode()` ayrıntı verir.

Çıktı (MySQL sürücüsü yokken)

```
java.sql.SQLException: No
suitable driver
found for
jdbc:mysql://localhost/okul
```

Bağlantı da bir kaynaktır; try-with-resources ile kapatılmalıdır.

KONTROL EDİLEN SINIFLAR

ParseException ve ClassNotFoundException

`ParseException` (`java.text`)

Bir metin belirli bir **biçime** göre ayrıştırılırken uyuşmazlık olursa fırlatılır.

```
try {
  SimpleDateFormat form =
    new SimpleDateFormat("dd/MM/yy");
  Date date = form.parse("01-01-2022");
} catch (ParseException e) {
  // Ayrıştırma hatasıyla ilgili işlemler
  System.out.println(e);
}
```

Çıktı

```
java.text.ParseException: Unparseable date:
"01-01-2022"
```

Biçim `/` bekliyor, metinde `-` var.

`ClassNotFoundException`

Bir sınıf **adıyla** yüklenmek istenirken (`Class.forName`) sınıf bulunamazsa fırlatılır.

```
try {
  Class.forName("java.util.Scanner");
  System.out.println("Scanner yüklendi");
  // Var olmayan bir sınıfı yüklemeye çalışma
  Class.forName("com.example.NonExistentClass");
} catch (ClassNotFoundException e) {
  System.out.println(e);
}
```

Çıktı

```
Scanner yüklendi
java.lang.ClassNotFoundException:
  com.example.NonExistentClass
```

KONTROL EDİLEN SINIFLAR

NotSerializableException

```
class Nokta {                // Serializable değil
  int x = 3, y = 4;
}
// main içinde
try (ObjectOutputStream outputStream =
    new ObjectOutputStream(new ByteArrayOutputStream())) {
  // Nesneyi ObjectOutputStream ile serileştirmeye çalışma
  Nokta notSerializableObj = new Nokta();
  outputStream.writeObject(notSerializableObj);
} catch (NotSerializableException e) {
  System.out.println(e);
} catch (IOException e) {
  System.out.println("Diğer G/Ç hatası: " + e);
}
```

Çıktı

```
java.io.NotSerializableException: Nokta
```

- **Serileştirme** (serialization): bir nesnenin bayt akışına dönüştürülmesi (dosyaya yazmak, ağdan göndermek) ve daha sonra geri dönüştürülmesidir.
- `Serializable` arayüzünü uygulayan sınıf serileştirilebilir kabul edilir; uygulamayan sınıfın nesnesi yazılmak istenince bu istisna oluşur.
- `IOException` 'ın (dolaylı) alt sınıfıdır; bu yüzden catch'i `IOException` 'dan **önce** yazılır.

✓ ÇÖZÜM

```
class Nokta implements Serializable {
  int x = 3, y = 4;
}
```

Error Sınıfları

- Programın normal çalışmasını **temelden** etkileyen, **sistem düzeyindeki** sorunları temsil eden kontrol edilmeyen tiplerdir.
- Genellikle uygulama kodunun bir satırından değil, JVM'in ve ortamın durumundan kaynaklanır.
- JVM tarafından fırlatılır; ele alınması veya bildirilmesi **önerilmez**. Programın düzgün biçimde sonlanması daha uygun bir yaklaşımdır.
- Yine de `Throwable` oldukları için teknik olarak `catch` edilebilirler.

Hata	Ne zaman?
`OutOfMemoryError`	Bellek (heap) tükendi
`StackOverflowError`	Çağrı yığını taştı: genellikle sonsuz özyineleme
`AssertionError`	`assert` koşulu yanlışsa (yalnızca `-ea` açıkken)
`NoClassDefFoundError`	Derlemede var olan sınıf çalışırken bulunamadı



Error Örnekleri

```java
static void sonsuz() {
  sonsuz();          // durma koşulu yok
}
public static void main(String[] args) {
  try {
    sonsuz();
  } catch (StackOverflowError e) {
    System.out.println("SOE -> " + e);
  }
  try {
    long[] dizi = new long[Integer.MAX_VALUE];
  } catch (OutOfMemoryError e) {
    System.out.println("OOM -> " + e);
  }
  int yas = -5;
  assert yas >= 0 : "Yaş negatif: " + yas;
  System.out.println("assert geçti");
}
```

java Ornek

```
SOE -> java.lang.StackOverflowError
OOM -> java.lang.OutOfMemoryError:
Requested array size exceeds VM limit
assert geçti
```

java -ea Ornek

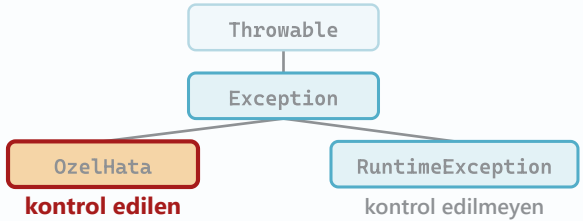
```
SOE -> java.lang.StackOverflowError
OOM -> java.lang.OutOfMemoryError:
Requested array size exceeds VM limit
Exception in thread "main"
java.lang.AssertionError: Yaş negatif: -5
```

- Özyinelemede taban durum unutulursa her çağrı yeni bir çerçeve ekler; yığın dolar.
- `assert` varsayılan olarak **kapalıdır**; `-ea` ile açılır. Geliştirirken varsayımları sınamak içindir. (Burada yakalamak yalnızca gösterim içindir.)

Özel İstisna Sınıfları

```java
public class OzelHata extends Exception {
  // Constructor
  public OzelHata(String message) {
    super(message);
  }
}
```

Üst sınıf	Sonuç
`extends Exception`	Kontrol edilen: çağıran ele almak zorunda; kurtarılabilir iş kuralı ihlalleri
`extends RuntimeException`	Kontrol edilmeyen: hatalı kullanım, programcı hatası



- Geliştirici, kendi programında ortaya çıkabilecek beklenmeyen durumları yönetmek için **özel istisna sınıfları** oluşturabilir.
- `Throwable`'ı genişleten her sınıfın nesnesi fırlatılabilir. Ancak genellikle `Exception` veya `RuntimeException` genişletilir; `Error` ve `Throwable` doğrudan genişletilmez.
- Yapıcı, mesajı `super(message)` ile üst sınıfa iletir; böylece `getMessage()` çalışır.
- Ad `…Hata` / `…Exception` ile bitirilir; durumu anlatmalıdır.
- Ek alanlar (eksik tutar, hatalı değer) eklenebilir: istisna sıradan bir sınıftır.

Özel İstisnayı Fırlatmak ve Yakalamak

```java
public class OzelHataTest {
  public static void main(String[] args) {
    try {
      fonksiyon();
    } catch (OzelHata e) {
      System.out.println("Yakalandı: " + e.getMessage());
    }
  }
  public static void fonksiyon() throws OzelHata {
    if (true) {
      throw new OzelHata("Özel hata oluştu.");
    }
  }
}
```

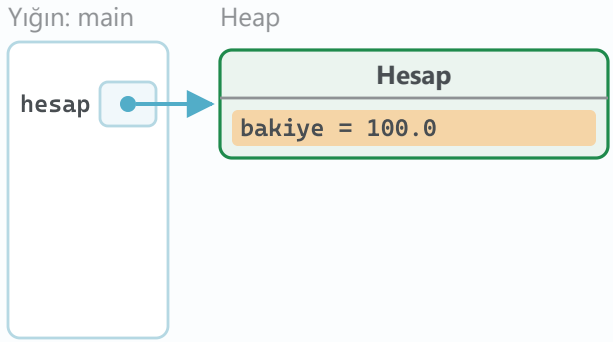
Çıktı

```
Yakalandı: Özel hata oluştu.
```

- `OzelHata`, `Exception`'dan türediği için kontrol edilen bir istisnadır: `fonksiyon()` imzasında `throws OzelHata` **zorunludur**.
- `main`, çağrıyı `try` içine alarak "ele al" seçeneğini kullanır.
- `if (true)` burada yalnızca gösterim içindir: koşulsuz `throw`'dan sonra gelen bir kod "unreachable statement" hatası verirdi.
- Gerçek programlarda koşul bir **iş kuralını** denetler: bakiye yeterli mi, not 0–100 arasında mı, kullanıcı adı boş mu...

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
7        throw new YetersizBakiyeHatasi(
8          "Bakiye yetersiz", miktar - bakiye);
9      }
10     bakiye -= miktar;
11     System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
▶ Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
20 } catch (YetersizBakiyeHatasi e) {
21   System.out.println(e.getMessage()
22     + ", eksik: " + e.getEksik());
23 }
```



Hesap nesnesi heap'te oluşur; `hesap` onu gösterir.

Adım 1 / 7

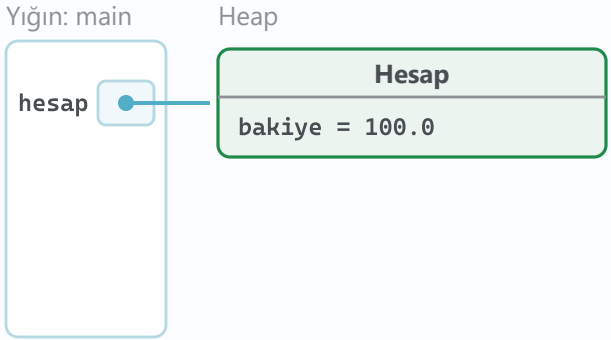
ÇAĞRI YIĞINI

`main()`

ÇIKTI

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
7        throw new YetersizBakiyeHatasi(
8          "Bakiye yetersiz", miktar - bakiye);
9      }
10     bakiye -= miktar;
11     System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
15 Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
20 } catch (YetersizBakiyeHatasi e) {
21   System.out.println(e.getMessage()
22     + ", eksik: " + e.getEksik());
23 }
```



`paraCek(30)` : 30 > 100 mü? **Hayır**; istisna yok.

Adım 2 / 7

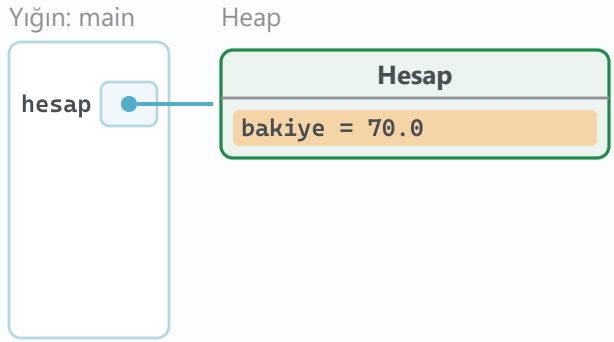
ÇAĞRI YIĞINI

- `paraCek(30.0)`
- `main()`

ÇIKTI

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
7        throw new YetersizBakiyeHatasi(
8          "Bakiye yetersiz", miktar - bakiye);
9      }
▶      bakiye -= miktar;
▶      System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
15 Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
20 } catch (YetersizBakiyeHatasi e) {
21   System.out.println(e.getMessage()
22     + ", eksik: " + e.getEksik());
23 }
```



Bakiye 70.0 olur ve yazdırılır.

Adım 3 / 7

ÇAĞRI YIĞINI

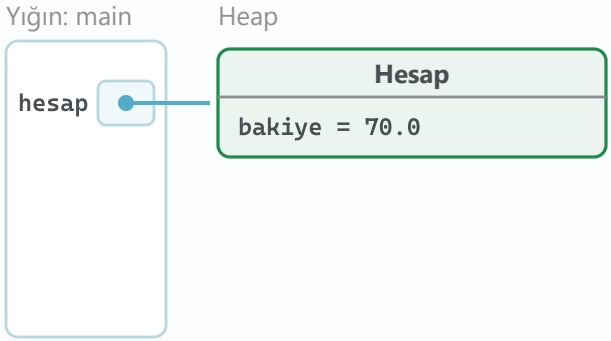
- `paraCek(30.0)`
- `main()`

ÇIKTI

```
Kalan: 70.0
```

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
7        throw new YetersizBakiyeHatasi(
8          "Bakiye yetersiz", miktar - bakiye);
9      }
10     bakiye -= miktar;
11     System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
15 Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
20 } catch (YetersizBakiyeHatasi e) {
21   System.out.println(e.getMessage()
22     + ", eksik: " + e.getEksik());
23 }
```



`paraCek(90)` : 90 > 70 mi? **Evet**.

Adım 4 / 7

ÇAĞRI YIĞINI

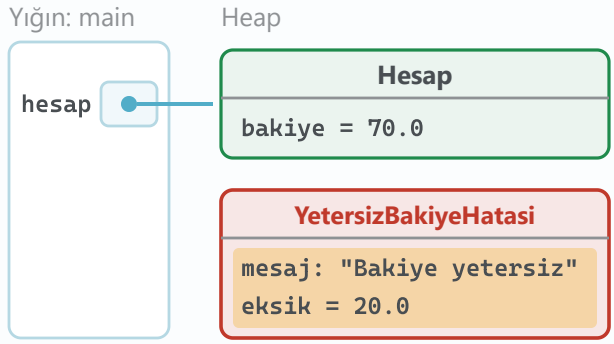
- **paraCek(90.0)**
- main()

ÇIKTI

```
Kalan: 70.0
```

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
▶        throw new YetersizBakiyeHatasi(
▶          "Bakiye yetersiz", miktar - bakiye);
9      }
10     bakiye -= miktar;
11     System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
15 Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
20 } catch (YetersizBakiyeHatasi e) {
21   System.out.println(e.getMessage()
22     + ", eksik: " + e.getEksik());
23 }
```



Heap'te bir **istisna nesnesi** oluşur (mesaj ve eksik = 20.0) ve fırlatılır. Bakiye **değişmez**.

Adım 5 / 7

ÇAĞRI YIĞINI

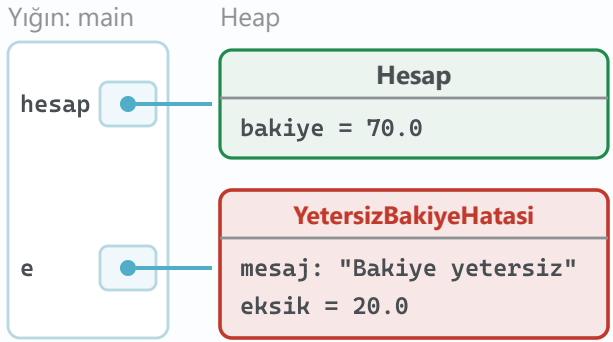
- paraCek(90.0)
- main()

ÇIKTI

```
Kalan: 70.0
```

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
7        throw new YetersizBakiyeHatasi(
8          "Bakiye yetersiz", miktar - bakiye);
9      }
10     bakiye -= miktar;
11     System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
15 Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
▶ } catch (YetersizBakiyeHatasi e) {
21   System.out.println(e.getMessage()
22     + ", eksik: " + e.getEksik());
23 }
```



paraCek'in çerçevesi atılır; `paraCek(10)` **atlanır**. catch eşleşir: `e` aynı nesneyi gösterir.

Adım 6 / 7

ÇAĞRI YIĞINI

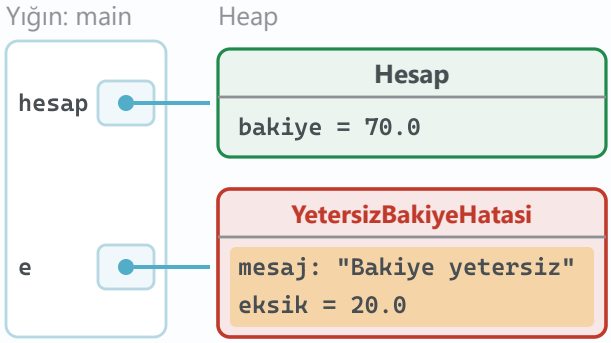
`main()`

ÇIKTI

```
Kalan: 70.0
```

Özel İstisna: Banka Hesabı

```
1  class Hesap {
2    private double bakiye;
3    Hesap(double bakiye) { this.bakiye = bakiye; }
4    public void paraCek(double miktar)
5        throws YetersizBakiyeHatasi {
6      if (miktar > bakiye) {
7        throw new YetersizBakiyeHatasi(
8          "Bakiye yetersiz", miktar - bakiye);
9      }
10     bakiye -= miktar;
11     System.out.println("Kalan: " + bakiye);
12   }
13 }
14 // main içinde
15 Hesap hesap = new Hesap(100);
16 try {
17   hesap.paraCek(30);
18   hesap.paraCek(90);
19   hesap.paraCek(10);
20 } catch (YetersizBakiyeHatasi e) {
▶    System.out.println(e.getMessage()
▶      + ", eksik: " + e.getEksik());
23 }
```



İstisna nesnesinin bilgileri kullanılır. İstisna da sıradan bir nesnedir.

Adım 7 / 7

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
Kalan: 70.0
Bakiye yetersiz,
eksik: 20.0
```

Bilgi Taşıyan Özel İstisna

```
class YetersizBakiyeHatasi extends Exception {
  private double eksik;

  public YetersizBakiyeHatasi(String mesaj, double eksik) {
    super(mesaj);          // mesaj Throwable'da saklanır
    this.eksik = eksik;
  }

  public double getEksik() {
    return eksik;
  }
}
```

- Önceki izlemede kullanılan sınıf: mesaja ek olarak **eksik tutarı** da taşır.
- Yakalayan kod bu bilgiyle anlamlı bir tepki verebilir: "20 TL eksik; yine de denemek ister misiniz?"
- `-1` gibi bir dönüş kodu bu kadar bilgi taşıyamazdı.
- Kontrol edilen tür seçildi: yetersiz bakiye **öngörülebilir** bir durumdur ve çağıran kod bununla başa çıkmalıdır.

Yığın İzi (Stack Trace)

- Program çalışırken **hangi metot çağrılarının** gerçekleştiğini ve istisnanın **nerede** oluştuğunu gösteren rapordur.
- İstisnanın tipini ve mesajını; her çağrı için sınıf ve metot adını, dosya adını ve satır numarasını içerir.
- Yakalanmayan bir istisnada JVM tarafından konsola (`System.err`) yazdırılır; `e.printStackTrace()` ile istenildiğinde de yazdırılabilir.
- Programcının hatayı anlamasına ve sorunu çözmesine yardım eden **en önemli** ipucudur.

```
Exception in thread "main" java.lang.NullPointerException
    at com.example.MyClass.myMethod(MyClass.java:10)
    at com.example.MyClass.main(MyClass.java:6)
```

Okuma: `main` (6. satır) `myMethod` 'u çağırdı; istisna `myMethod` 'un 10. satırında oluştu. `com.example` paket, `MyClass` sınıf adıdır.

Yığın İzini Okumak

```java
public class IzlemeOrnegi {
  static int uzunluk(String metin) {
    return metin.length();
  }
  static void yazdir(String ad) {
    System.out.println(ad + ": " + uzunluk(ad));
  }
  public static void main(String[] args) {
    yazdir("Ali");
    yazdir(null);
    System.out.println("Bitti");
  }
}
```

1. **İlk satır**: tip ve mesaj: `metin` `null`.
2. **İlk `at`** : istisnanın **doğduğu** yer (3. satır).
3. Aşağı doğru çağrı zinciri: `main` (10) → `yazdir` (6) → `uzunluk` (3).
4. Önce **programın kendi** satırlarına bakılır; asıl neden çoğu zaman aşağıdadır: `null` 'ı 10. satır gönderdi.

Çıktı

```
Ali: 3
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because
"metin" is null
    at IzlemeOrnegi.uzunluk(IzlemeOrnegi.java:3)
    at IzlemeOrnegi.yazdir(IzlemeOrnegi.java:6)
    at IzlemeOrnegi.main(IzlemeOrnegi.java:10)
```

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
3      return metin.length();
4    }
5    static void yazdir(String ad) {
6      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
▶      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Çağrı yığını (sağdaki en üstte)

`main`

main 9. satırda `yazdir("Ali")` 'yı çağırır.

Adım 1 / 9

ÇIKTI

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
3      return metin.length();
4    }
5    static void yazdir(String ad) {
▶      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Çağrı yığını (sağdaki en üstte)



yazdir 6. satırda `uzunluk(ad)` 'ı çağırır. Her çerçeve, bulunduğu satırı hatırlar.

Adım 2 / 9

ÇIKTI

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
▶      return metin.length();
4    }
5    static void yazdir(String ad) {
6      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Çağrı yığını (sağdaki en üstte)



`"Ali".length()` = 3 döner; çerçeve silinir.

Adım 3 / 9

ÇIKTI

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
3      return metin.length();
4    }
5    static void yazdir(String ad) {
▶      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Çağrı yığını (sağdaki en üstte)



"Ali: 3" yazılır, yazdir biter.

Adım 4 / 9

ÇIKTI

```
Ali: 3
```

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
3      return metin.length();
4    }
5    static void yazdir(String ad) {
6      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
▶      yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Çağrı yığını (sağdaki en üstte)

`main`

main 10. satırda `yazdir(null)` 'ı çağırır.

Adım 5 / 9

ÇIKTI

```
Ali: 3
```

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
3      return metin.length();
4    }
5    static void yazdir(String ad) {
▶      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Çağrı yığını (sağdaki en üstte)



`null` parametre olarak iletilir; henüz sorun yok.

Adım 6 / 9

ÇIKTI

```
Ali: 3
```

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
▶      return metin.length();
4    }
5    static void yazdir(String ad) {
6      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```



`null.length()` : NPE fırlatılır. O anda yığında üç çerçeve var: uzunluk:3, yazdir:6, main:10.

ÇIKTI

```
Ali: 3
```

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
▶      return metin.length();
4    }
5    static void yazdir(String ad) {
6      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
10     yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Yığın izi = yığının fotoğrafı

```
at uzunluk(IzlemeOrnegi.java:3)
at yazdir(IzlemeOrnegi.java:6)
at main(IzlemeOrnegi.java:10)
```

İstisna nesnesi doğduğu anda yığının kaydını alır: her çerçeve bir `at` satırı olur, en üstteki ilk sırada.

Adım 8 / 9

ÇIKTI

```
Ali: 3
```

Yığın İzi Nereden Gelir?

```
1  public class IzlemeOrnegi {
2    static int uzunluk(String metin) {
3      return metin.length();
4    }
5    static void yazdir(String ad) {
6      System.out.println(ad + ": " + uzunluk(ad));
7    }
8    public static void main(String[] args) {
9      yazdir("Ali");
▶      yazdir(null);
11     System.out.println("Bitti");
12   }
13 }
```

Yığın izi = yığının fotoğrafı

```
at uzunluk(IzlemeOrnegi.java:3)
at yazdir(IzlemeOrnegi.java:6)
at main(IzlemeOrnegi.java:10)
```

Kimse yakalamadı: üç çerçeve de atıldı, JVM yığın izini yazıp programı sonlandırdı. “Bitti” yazılmaz.

Adım 9 / 9

ÇIKTI

```
Ali: 3
Exception in thread "main"
java.lang.NullPointerException: Cannot invoke
"String.length()" because "metin" is null
    at IzlemeOrnegi.uzunluk(IzlemeOrnegi.java:3)
    at IzlemeOrnegi.yazdir(IzlemeOrnegi.java:6)
    at IzlemeOrnegi.main(IzlemeOrnegi.java:10)
```

YIĞIN İZİ

İstisna Yönetiminde İyi Uygulamalar

Özel tipi yakalamak

Mümkünse `catch (Exception e)` yerine beklenen tip yakalanır; her türe uygun tepki verilir.

İstisnayı yutmamak

Boş `catch { }` hatayı gizler. En azından kayda geçirilir (log) veya kullanıcıya bildirilir.

Kaynakları kapatmak

Dosya, bağlantı gibi kaynaklar try-with-resources veya `finally` ile kapatılır.

Önlenebilecekleri önlemek

`null`, sınır ve sıfır denetimi yapılır; `RuntimeException` yakalamak yerine kod düzeltilir.

Akış denetimi için kullanmamak

Döngüden çıkmak gibi olağan işler istisnayla yapılmaz; istisnalar **istisnai** durumlar içindir.

Anlamlı mesaj

Mesaj hatalı değeri ve beklentiyi söyler: `"Not 0–100 arası olmalı: 120"`.

Doğru yerde ele almak

İstisna, çözülebileceği yerde yakalanır; çözülemiyorsa `throws` ile iletilir.

Kayıt tutmak

İstisnaları bir dosyaya kaydetmek (logging), sonradan hatanın izini sürmeyi kolaylaştırır.

YIĞIN İZİ

final, finally ve finalize

	Ne?	Görevi
`final`	Anahtar sözcük	Değişken: yeniden atanamaz · metot: geçersiz kılınamaz · sınıf: alt sınıfı olamaz
`finally`	Anahtar sözcük (blok)	`try` 'dan sonra, istisna olsun olmasın her durumda çalışan temizlik bloğu
`finalize()`	`Object` sınıfının metodu	Çöp toplayıcı nesneyi silmeden önce çağırabilirdi. Ne zaman çağrılacağı belirsizdir; Java 9'dan beri **kullanımdan kaldırılmış** (deprecated) durumdadır.

Adları benzer, işleri tamamen farklıdır. Kaynak temizliği için `finalize()` değil, **try-with-resources** veya `finally` kullanılır.

Özet

Kavram	Özü
İstisna	Çalışma zamanında normal akışı bozan durum; tipi, mesajı ve yeri olan bir nesne
`try` / `catch`	Riskli kod `try` 'da; eşleşen **ilk** catch çalışır, try'ın kalanı atlanır
catch sırası	Alt sınıf önce, üst sınıf sonra; tersi derleme hatası. `A | B` (Java 7+)
`finally`	Her durumda çalışır, `return` 'den bile önce; temizlik için
try-with-resources	`try (Kaynak k = …)` : `AutoCloseable` kaynak otomatik kapanır
Hiyerarşi	`Throwable` → `Error` | `Exception` → … `RuntimeException`
Kontrol edilen	`Exception` (Runtime hariç): ele al **veya** `throws` ile bildir
Kontrol edilmeyen	`RuntimeException` , `Error` : zorunluluk yok; önle
`throw` / `throws`	Fırlat (gövdede, nesne) / bildir (imzada, tipler)
Yayılım	Yakalanmayan istisna çağıran metoda geçer; `main` de yakalamazsa program sonlanır
Özel istisna	`class X extends Exception` + `super(mesaj)`
Yığın izi	Tip + mesaj, sonra çağrı zinciri: ilk `at` satırı istisnanın doğduğu yer

ÖZET

Sık Yapılan Hatalar

Kontrol edilen istisnayı unutmak
`new FileReader(ad)`, try/throws yok: `unreported exception FileNotFoundException; must be caught or declared to be thrown`.

Genel catch'i öne yazmak
`catch (Exception e)`, sonra `catch (ArithmeticException e)`: `exception ArithmeticException has already been caught`.

Fırlatılmayanı yakalamak
Dosya işlemi olmayan try'a `catch (FileNotFoundException e)`: `exception FileNotFoundException is never thrown in body of corresponding try statement`.

Yalnız try yazmak
catch veya finally olmayan try: `'try' without 'catch', 'finally' or resource declarations`.

throw'dan sonra kod yazmak
Aynı blokta `throw`'un ardından gelen satır: `unreachable statement`.

try içindeki değişkeni dışarıda kullanmak
try içinde tanımlanan `sonuc` sonra yazdırılırsa: `cannot find symbol`. Değişken try'dan önce tanımlanmalıdır.

İstisnayı yutmak
`catch (Exception e) { }` derlenir ama hata iz bırakmadan kaybolur.

Metin fırlatmak
`throw "hata";` → `incompatible types: String cannot be converted to Throwable`.

ÖZET

Örnek Problemler (1)

PROBLEM 1: ÇIKTI NEDİR?

```
static int f() {
  try {
    System.out.print("A");
    int x = 5 / 0;
    System.out.print("B");
    return 1;
  } catch (ArithmeticException e) {
    System.out.print("C");
    return 2;
  } finally {
    System.out.print("D");
  }
}
// main içinde
System.out.println(" " + f());
```

PROBLEM 2: ÇIKTI NEDİR?

```
try {
  String s = "12a";
  int n = Integer.parseInt(s);
  System.out.println("Sayı: " + n);
} catch (IllegalArgumentException e) {
  System.out.println("IAE");
} catch (Exception e) {
  System.out.println("Exception");
}
```

PROBLEM 3

Problem 2'de iki catch bloğunun yeri değiştirilirse ne olur?

▶ **Cevapları göster**

Örnek Problemler (2)

PROBLEM 4: ÇIKTI NEDİR?

```
static void a() {
  System.out.print("a1 ");
  b();
  System.out.print("a2 ");
}
static void b() {
  System.out.print("b1 ");
  throw new IllegalStateException("!");
}
// main içinde
try {
  a();
  System.out.print("m ");
} catch (IllegalStateException e) {
  System.out.print("yakalandı" + e.getMessage());
} finally {
  System.out.println(" son");
}
```

İstisna `b()`'de doğuyor; `b()` ve `a()`'da try yok. Hangi `print`'ler çalışır, hangileri atlanır? `finally` ne zaman çalışır?

▶ **Cevabı göster**

ÖZET

Örnek Problemler (3)

PROBLEM 5: TOPLAM KAÇTIR?

```
int[] d = {4, 0, 2};
int toplam = 0;
for (int i = 0; i <= d.length; i++) {
  try {
    toplam += 12 / d[i];
  } catch (ArithmeticException e) {
    toplam += 100;
  } catch (ArrayIndexOutOfBoundsException e) {
    toplam += 1000;
  }
}
System.out.println(toplam);
```

PROBLEM 6: ÇIKTI NEDİR?

```
try {
  throw new RuntimeException("X");
} finally {
  System.out.println("F");
}
```

▶ **Cevapları göster**

ÖZET

Alıştırmalar (1)

1. Metni yaşa çeviren `int yasOku(String metin)` yazın: sayı değilse `NumberFormatException` doğal olarak oluşsun, negatifse `IllegalArgumentException` fırlatın. `"20"`, `"-3"`, `"yirmi"` girdileri için iki catch ile mesaj yazdırın.
2. `GecersizNotHatasi` adlı kontrol edilen bir istisna ve not 0–100 dışındaysa onu fırlatan `notGir(int not)` yazın.
3. 1. alıştırmada catch sırası `IllegalArgumentException`, sonra `NumberFormatException` olursa ne olur?

▶ **Cevaplar / ipuçları**

ÖZET

Alıştırmalar (2)

4. Sınıflandırın (kontrol edilen / edilmeyen): `IOException`, `NumberFormatException`, `SQLException`, `NullPointerException`, `FileNotFoundException`, `ClassCastException`, `StackOverflowError`.
5. Bir `BufferedReader`'ı `finally` ile kapatan kodu try-with-resources ile yeniden yazın.
6. `double ortalama(int[] notlar)` metodu boş dizi için ne yapmalı? `0` mı döndürmeli, istisna mı fırlatmalı? Hangi istisna?

▶ **Cevaplar / ipuçları**

ÖZET

Kaynaklar

- Oracle, *The Java Tutorials*: Lesson: Exceptions (Catching and Handling Exceptions; The try-with-resources Statement; Unchecked Exceptions — The Controversy).
- J. Gosling ve diğ., *The Java Language Specification*, Java SE 21 Edition, Bölüm 11: Exceptions; 14.20 The try statement.
- Oracle, *Java SE 21 API Documentation*: `java.lang.Throwable`, `Exception`, `RuntimeException`, `Error`.
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson (Bölüm 11: Exception Handling: A Deeper Look).
- K. Sierra, B. Bates ve T. Gee, *Head First Java*, 3. baskı, O'Reilly (Risky Behavior: exception handling).
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson (Bölüm 7: Exceptions, Assertions, and Logging).
- J. Bloch, *Effective Java*, 3. baskı, Addison-Wesley (Bölüm 10: Exceptions, Madde 69–77).

Son

Bölüm 6: İstisnalar

Sonraki bölüm: Bölüm 7: Test ve Hata Ayıklama · Terim sözlüğü