İstisna Yönetimi
(Exception Handling)

Computer Engineering Department

Object Oriented Programming Course

Prof. Dr. Ahmet Sayar

Kocaeli University - Fall 2023

Içerik

- İstisna Yönetimi (Exception Handling)
  - İstisna Nedir?
  - İstisna Yakalama Mekanizması
  - İstisna İşleme Modeli
  - İstisnaların Listesi
  - İstisna Fırlatma
  - İstisna Oluşturma

İstisna (exception) Nedir?

- İstisna, bir kod dizisinde, çalışma sırasında ortaya çıkan, anormal bir durumdur.
- Bir çalışma zamanı (run-time) hatasıdır.
- İstisnalar, programcılara, hatalar karşısında istenildiği şekilde davranabilme ve oluşacak hataları kontrol altına alabilme yeteneği ve esnekliği sağlar.

İstisna (exception) Nedir?

- Bir sistemi tamamen istisnalara karşı korunaklı tasarlamak ve oluşturmak çok zordur.
- Çıkabilecek sorunların önceden tamamıyla düşünülmesi mümkün değildir.
- Sorunlar ciddi farklılıklar gösterebilir.
  - Veri tutarsızlığı
  - Operatör hatası
  - Bellek hatası
  - Yanlış girdi
  - Donanımsal tutarsızlık
  - …
- Sistemlerin tamamen çökmesinin veya güvenlik problemleriyle karşılaşmalarının %80 oranında istisnalardan kaynaklandığı bilinmektedir.

İstisna yöneticisi (exception handler)

- Yazmakta olduğumuz bir programda yüzden fazla yerde dosya sonu problemi ve bir o kadar da sıfıra bölme problemi olduğunu düşünelim.
  - Yapısal dillerde tüm bu problemler algoritma içerisine gömülü olarak yazılan kod parçalarıyla kontrol altına alınmalıdır.
  - Bu da ikiyüz defa aynı kontrolün yazılması anlamına gelebilir.
  - Oysa istisna işleme sayesinde programın bu tip hatalarla karşılaştığında davranışının ne olacağı sadece bir istisna işleyici ile belirlenmektedir.
  - Bu istisna işleyici birden çok program birimi tarafından kullanılabilir; bu sayede kodun hantallaşması önlendiği gibi güvenilirlik azami şekilde sağlıklı kılınmaktadır.

İstisna yöneticisi (exception handler)

- Bir Java istisnası, kod parçasının çalışma zamanında meydana gelen istisnai bir durumu tarif eden nesnedir.

- Ana mantık şu şekildedir:
  - İstisnai bir durum ortaya çıktığında, o istisnayı temsil eden bir nesne yaratılır ve hataya sebep olan metodun içine atılır (*throw).*
  - Programcının olaya nasıl bakması gerektiğiyle ilgili olarak bu metot istisnayı ya yakalar (*catch) ve işler yada yakalanıp işlenmesi için istisnaya dokunmaz ve geçer.*

- Java'da istisna işleme konusunda beş anahtar sözcük karşımıza çıkmaktadır:
  - *try*
  - *catch*
  - *throw*
  - *throws*
  - *finally*

İstisna Bloku

```
try {
        code();
}
catch (exception 1) {
        NecessaryErrorProcess1();
}
...
catch (exception N) {
        NecessaryErrorProcessN();
}

finally {
        Finally();
}
```

İstisna Bloku

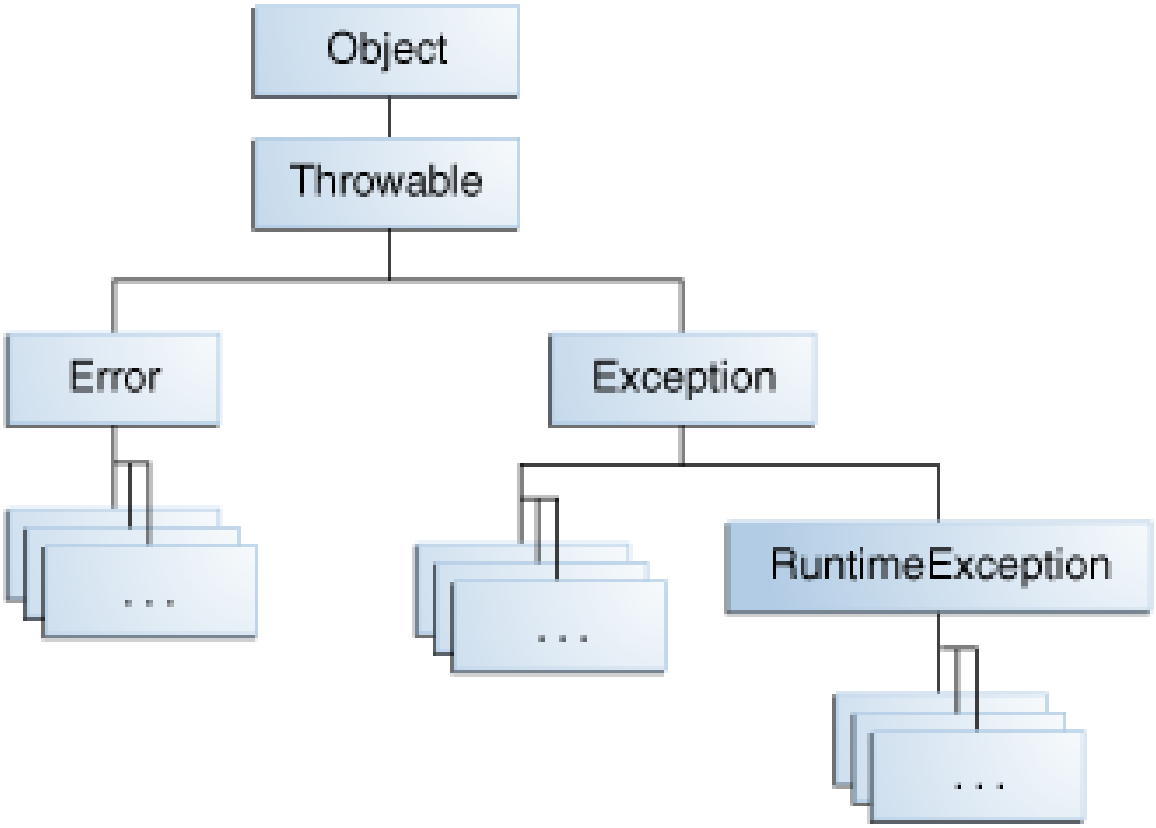
- İstisnalar için izlenen kod, bir *try bloğu içerisinde tutulur.*
  - Bu bloğun anlamı: ‘bu kodu çalıştır ve istisna oluşacak mı gör‘
- Eğer try bloğu içerisinde bir istisna meydana gelirse, bir istisna nesnesi yaratılır ve kod içerisine atılır.
- Programımız bu istisnayı *catch ifadesi ile yakalayabilir ve işleyebilir.*
  - Her catch bloğu ne tipte bir istisnayı yakalayabileceğini belirler ve içerisinde o tipte istisnayı işlemek üzere bir istisna işleyici bulunur.
- Eğer programcı bir istisnayı bizzat atmak istiyorsa *throw ifadesini kullanır.*
- Bir metottan atılan her istisna bir *throws kalıbı ile belirtilir.*
- *Finally ile belirtilen bloğun içindeki kod, istisna oluşsun oluşmasın kesinlikle çalışması gereken koddur.*
  - Bazı kodların her ne olursa olsun çalışması gerekiyorsa *finally bloğunun içinde yer almalıdır.*

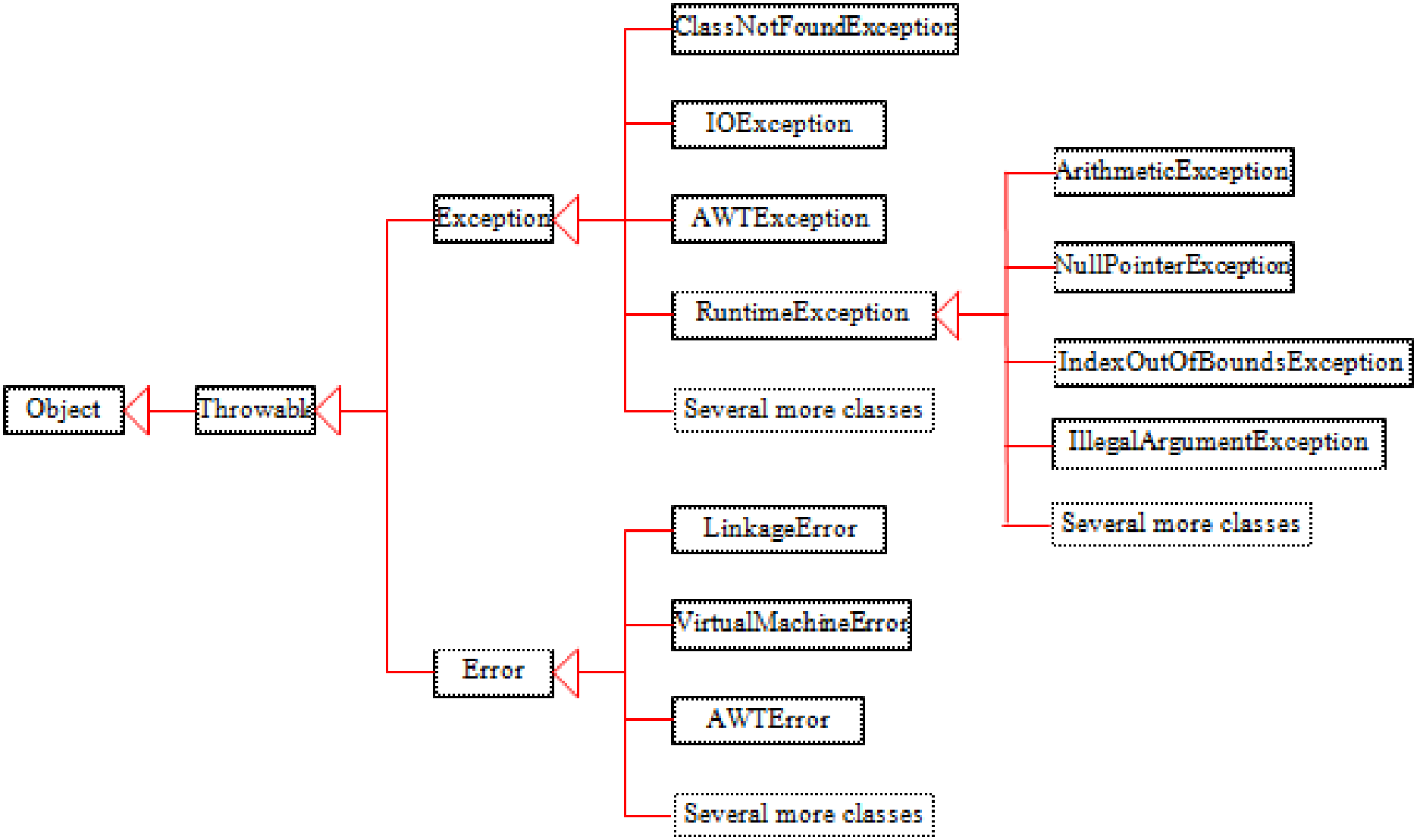
Terminoloji

- Çalışma zamanında beklenmeyen bir problem oluşması durumu → exception
- Bir istisnanın yaratılıp program içerisine atılması → throwing
- Oluşan bir istisnayı yakalayıp problemin çözülmesi adına belirtilen ifadeleri çalıştırma işlemi → catching
- İstisna ile uğraşan kod bloğu →catch clause (catch block)
- İstisnanın oluştuğu noktada kontrol sağlayan bir dizi metot çağırma işlemi → stacktrace

Adı	Açıklaması
RuntimeException	Pek çok java.lang istisnalarının temel sınıfı
ArithmeticException	Aritmetik hata
IllegalAccessException	Sınıf erişilebilir değil
IllegalArgumentException	Metot geçersiz argüman yürüttü
ArrayIndexOutOfBoundsException	Dizi boyutu sıfırdan küçük veya asıl dizi boyutundan büyük
NullPointerException	'null' nesne üyesi erişim denemesi
SecurityException	Güvenlik ayarları işleme izin vermiyor
ClassNotFoundException	İstenen sınıf yüklenemiyor
NumberFormatException	Dizgiden kayan noktaya geçersiz çevirme
AWTException	AWT'de istisna
IOException	Giriş/Çıkış istisnaları için temel sınıf
FileNotFoundException	Dosya bulunamadı
EOFException	Dosya sonu
NoSuchMethodException	İstenilen metod mevcut değil
InterruptedException	İşletim dizisi durduruldu
RuntimeException	Pek çok java.lang istisnalarının temel sınıfı
ArithmeticException	Aritmetik hata
IllegalAccessException	Sınıf erişilebilir değil
IllegalArgumentException	Metot geçersiz argüman yürüttü
ArrayIndexOutOfBoundsException	Dizi boyutu sıfırdan küçük veya asıl dizi boyutundan büyük

Sınıf Hiyerarşisi




Object
Throwable
Exception
Error
ClassNotFoundException
IOException
AWTException
RuntimeException
Several more classes
ArithmeticException
NullPointerException
IndexOutOfBoundsException
IllegalArgumentException
Several more classes
LinkageError
VirtualMachineError
AWTError
Several more classes

Exception and Error

- Tüm istisna tipleri, yerleşik Throwable sınıfının alt sınıflarıdır.
- Throwable'ın hemen altında, istisnaları iki ayrı dala ayıran, iki alt sınıf vardır:

1. Exception
   - Kullanıcı programlarının yakalaması gereken istisnai durumlar için kullanılır.
   - Aynı zamanda kullanıcı tanımlı istisnalar için alt sınıflandırılan sınıftır.
   - Runtime Exception adlı çok önemli bir alt sınıfı mevcuttur.
     - Bu sınıf içerisinde bulunan istisna tipleri yazılan program içerisinde otomatik olarak tanımlanır.

2. Error
   - Normal şartlar altında program tarafından yakalanmayacak istisnaları tanımlayan sınıftır.
   - JVM tarafından atılır
   - Error tipindeki istisnalar, Java run-time ortamının kendisi ile ilgili hatalarını göstermek için Java run-time sistemi tarafından kullanılır.

Throwable Sınıfı

- Tüm istisnaların kalıtımlandığı Throwable sınıfı, oluşan istisna hakkında bilgi döndürmek için kullanılan, üç tane metot içerir:
  - getMessage() → istisna hakkında text olarak bir bilgi döndürür.
  - printStackTrace() → bu istisnaya kadarki çağırma yığınlarını – hangi metotlar çağrılmıştır bilgisini- döndürür.
  - getCause() → exception nedenini döndürür

Try ve catch

- Bir run-time hatasına karşın önlem almak ve bu hatayı işlemek için, izlenmesi istenilen kod try bloğu içerisine alınır.
- Try bloğunun hemen arkasından, yakalamak istenilen istisna tipini belirten, bir catch kalıbı konulur.
- Try ve catch beraber bir birim oluştururlar.

Try ve catch

- İstisnanın işleme kodu catch bloğu içerisine yazılır.
- Bir catch kalıbı, içiçe yuvalanmış try blokları hariç, başka bir try bloğunun attığı istisnayı yakalayamaz.
  - İyi yapılandırılmış catch bloklarının amacı, istisnai durumları çözmek ve hata olmamış gibi programın akışını devam etmesini sağlamaktır.
- Bir istisna atıldığı zaman, program kontrolü try bloğundan catch bloğuna geçer.
  - Herhangi bir metodun çağrılması gibi catch bloğunun çağrılması söz konusu değildir.
  - Çalışma hiç bir zaman catch bloğundan try bloğuna dönmez.
    - Catch bloğu çalıştıktan sonra, program, try/catch kalıbının hemen ardından gelen kod parçalarıyla devam eder.

throw ve throws

- Program sadece Javarun-time sistemi tarafından yaratılan istisnaları yakalamak zorunda değildir.
- Throw deyimi kullanılarak istisnayı programcının atması da mümkündür.
- Atılacak istisna Throwable sınıfının veya bu sınıfın alt sınıflarının bir nesnesi olmak zorundadır.
- Eğer bir metot, işleyemediği bir istisnaya neden oluyorsa, bu durumu mutlaka kendini çağıran diğer programlara bildirmelidir.
  - Bu, metodun içerisine bir throws kalıbı sokularak yapılabilir.
- Throws kalıbı bir metodun fırlatabileceği istisnaları listeler.
- Bu listenin dışında bir istisna ortaya çıkacak olursa çalışma zamanı hatasına yol açar.

Örnek - throw

```
class OrnekIstisnaThrowUygulamasi{
                public static void main(String args[])
                {
                        try{
                                throw new ArithmeticException("0'a bolme");
                        }
                        catch (ArithmeticException e){
                                System.out.println("istisnai durum= "+e);
                        }
                        System.out.println("throw islemi sonrasi mesaj");
        }
}
```

- Çıktı ne olur?
- Catch'de ArithmeticException yerine sadece Exception yazsaydık ne olurdu?

Throws

```
/** Set a new radius */
 public void setRadius(double newRadius)
     throws IllegalArgumentException {
   if (newRadius >= 0)
     radius =  newRadius;
   else
     throw new IllegalArgumentException(
       "Radius cannot be negative");
 }
```

Örnek - throws

```
public class ThrowsDeneme
{
    static void atmaOrnegi() throws ArithmeticException{
        System.out.print("firlatilan istisna: ");
        throw new ArithmeticException("deneme");
    }

    public static void main(String args[]){
        try{
            firlatmaOrnegi();
        }
        catch(ArithmeticException e){
            System.out.println("burada yakalandi => "+e);
        }
    }
}
```

- Çıktı ne olur?

Catching Exceptions - Hiyerarşi

```
try {
  statements;  //Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

Hiyerarşi hatası – En özelden en genele

```
byte x;
String fileName = "dasdads";
FileInputStream file;

try {
    file = new FileInputStream(fileName);
    x = (byte) file.read();
} catch ( IOException i) {
    i.printStackTrace();
} catch (FileNotFoundException  f)  {
    f.printStackTrace();
}
```

Throws and catching

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

```
void p2() throws IOException {

  ……..

}
```

```
public class Firlatim {
  public void cokCalis() throws Exception {
    try {
      throw new Exception("bir istisna"); // istisnanın oluşumu
    } catch(Exception ex) {
      System.out.println("cokCalis() istisna yakalandı: " + ex);
      throw ex;
    }
  }
  public void calis() throws Exception {
    try {
      cokCalis();
    } catch(Exception ex) {
      System.out.println("calis() istisna yakalandı: " + ex);
      throw ex;
    }
  }
  public void basla() {
    try {
      calis();
    } catch(Exception ex) {
      ex.printStackTrace();
    }
  }
  public static void main(String args[]) {
   Firlatim f = new Firlatim1();
   f.basla();
  }
}
```

istisna cokCalis() metodunun içerisinde oluşuyor.

Oluşan istisna catch mekanizması sayesinde yakalandıktan sonra bir üst sınıfa fırlatılıyor.

calis() metodunun içerisinde de aynı şekilde fırlatılan istisna, catch mekanizmasıyla yakalanıp tekrar bir üst kısıma fırlatılıyor.

basla() metoduna kadar gelen istisna nesnesi burada yakalanıp içerisinde saklı bulunan bilgiler printStackTrace() metoduyla ortaya çıkartılıyor.

```
cokCalis() istisna yakaladi: java.lang.Exception: Bir istisna
java.lang.Exception: Bir istisna
Calis() istisna yakaladi: java.lang.Exception: Bir istisna
        at Firlatim.cokCalis(Firlatim.java:13)
        at Firlatim.calis(Firlatim.java:21)
        at Firlatim.basla(Firlatim.java:29)
        at Firlatim.main(Firlatim.java:36)
```

finally

- Finally bloğu içerisindeki kod mutlaka çalışması gereken koddur.
- Neden böyle bir şeye gerek duyulur?
  - Istisna oluşması programın akışına yön değiştirten bir durumdur.
  - Bazı metotların işlemesi mutlak gerekli ise bu durum ciddi problemler yaratabilir. Bu tip problemlerin oluşmaması için finally bloğu kullanılır.
  - Örneğin açık kalmış dosyaların kapatılması gibi işlemler genelde finally bloğu içerisine yazılarak olası problemeler engellenmiş olur.

Örnek - finally

```
public class FinallyDeneme
{
    static void finalOrnegi() throws ArithmeticException{
        System.out.print("firlatilan istisna: ");
        throw new ArithmeticException("deneme");
    }

    public static void main(String args[]){
        try{
            finalOrnegi();
        }
        finally{
                System.out.println("finally mutlaka calisir!");
            }
    }
}
```

program çıktısı:

firlatilan istisna: finally mutlaka çalışır!

Exception in thread "main" java.lang.ArithmeticException: deneme

Try ifadesi içerisinde çağrılan metod bir istisna fırlatıyor ve bunu yakalayacak tanımlı bir catch ifadesi olmadığı için istisna yönetimi Java'nın run-time istisna yöneticisine düşüyor.

Try içerisinden sapma oluşmasına rağmen finally bloğu çalıştırılıp program öyle kesiliyor.

Örneğin Devamı

```
public static void main(String args[]) {
      try{
         finalornegi();
      }
      catch (Exception e){
         System.out.println("helooooooo");
      }
      finally{
         System.out.println("finally mutlaka calisir ");
      }
   }
   static void finalornegi() throws ArithmeticException{
      System.out.println("firlatilan istina: ");
      throw new ArithmeticException("deneme");
   }
```

Örnek Kodun Çıktıları

- Catch kullanılınca cıktı ne olur

```
firlatilan istina:
helooooooo
finally mutlaka calisir
```

- Catch kullanılmayınca cıktı ne olur

```
firlatilan istina:
finally mutlaka calisir
Exception in thread "main" java.lang.ArithmeticException: deneme
        at Main.finalornegi(Main.java:28)
        at Main.main(Main.java:16)
Java Result: 1
```

İstisna Oluşturma -Ornek

- İstisna oluşumuna en basit örnek olarak, yanlış kullanılmış dizi uygulamasını verebiliriz.
  - Java programlama dilinde dizilere erişim her zaman kontrollüdür.
  - Java programlama dilinde dizilerin içerisine bir eleman atmak istiyorsak veya var olan bir elemana ulaşmak istiyorsak, bu işlemlerin hepsi Java tarafından önce bir kontrolden geçirilir.
  - Amaç, güvenli bir dizi erişim mekanizmasına sahip olmaktır.

```
public class Dizi{
   public static void main(String args[]) {
     int sayi[] = {1, 2};
     for (int i=0 ; i < 3 ; i++) {
        System.out.println("** " + sayi[i]);
     }
     System.out.println("Bitti");
   }
}
```

```
** 1
** 2
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
```

İstisna Oluşma Sebepleri

- Açmak istediğiniz fiziksel dosya yerinde olmayabilir.
- Uygulamanıza kullanıcılar tarafında, beklenmedik bir girdi kümesi gelebilir.
- Ağ bağlantısı kopmuş olabilir.
- Yazmak istediğiniz dosya, başkası tarafından açılmış olduğundan yazma hakkınız olmayabilir.
- Kullanıcı yanlış veri girdisi

Checked Exception

- Java'da kod geliştirme esnasında **try-catch-finally** bloğu içerisinde yakalanan ve işlem yapılabilen exceptionlar
- **Checked Exception** alındığında çalışma anında düzeltme yapıp düzgün parametrelerle kodun çalışmasına devam edilebilir.
- yazdığınız kodda **Checked Exception** varsa IDE sizi uyarıyor ve bu fırlatılan exception' ı handle etmeniz yani yakalayıp gerekli işlemi yapmanız gerektiğini söylüyor.
  - İşte bu durumda yazılımcılar olarak bizler, ya exception' ın fırlatıldığı satırı try-catch bloğu içine alıyoruz ya da bir üst metoda throws ediyoruz.

Unchecked Exception - Runtime Exception

- Java'da yine kod geliştirme esnasında **try-catch-finally** bloğuyla yakalanabilir ve checked exception tipinde olduğu gibi istenildiği gibi işlem devam ettirilebilir.
- Try Catch bloğu içinde yazılması gerektiğini kodcunun öngörmesi beklenir.
  - Kodu geliştirme esnasında IDE yazılımcıyı uyarmaz. Yani orada bir hata olacağı geliştirme esnasında değil çalışma esnasında ortaya çıkar.
  - ArrayIndexOutOfBoundException
- Bir diğer unchecked exception türü ise **Error** sınıflarıdır.
  - Kod geliştirme esnasında veya çalışma esnasında yazılımcının yakalayamacağı istisnalardır.
  - Problemin diğer Exception türlerine göre daha ciddi olduğu durumları yansıtmaktadır.
  - OutOfMemoryError

Hata Yönetimi vs. İstisna Yönetimi

- Aslında *Error sınıfından oluşan nesnelerin esas olarak yaptıkları şey programın akışını durdurup, sorunu belirtmek ve programdan çıkmaktır.*
  - Programcının bu hataları işlemesine gerek kalmadan sistem zaten bu hataları ele almaktadır.

- Programcının asıl yapması gereken iş çeşitli istisnaları sorun olmaktan çıkarmaya uğraşmaktır.

- Çalışma zamanı istisnaları genel olarak kod yazılımının yol açtığı hatalardır. Bu yüzden, programcı kendi hatalarını ele almasını sağlayacak istisna işleme teknikleri yaratmalıdır.

Hata (Error) vs. İstisna (Exception)

- Error, sistem bütünlüğünde, programcının kontrolü dışında gerçekleşen hatalardır.
  - Sistem içi hatalardan kastedilen programcının kontrolü dışında oluşan sorunlardır.
  - bellek yetersizliği, İnternet bağlantısının kurulamaması, disk problemleri, …
- İstisna ise çalışma zamanında meydana gelen, beklenmeyen durumlar olarak tanımlanır.
  - İstisnalar, genel olarak kod yazılması sırasında programcı tarafından yapılabilecek hatalardır.
  - Bu tanım, bütün istisnai durumların programcının yanlış söz dizileri kullanmasından kaynaklandığı anlamına gelmemelidir.
  - Doğrudan programcıyı ilgilendiren durumlardan bahsedilmektedir.
    - Yazılan programın beklediği girdiler, sunduğu çıktılar,…

Özet

- İstisna yönetiminin faydaları:
  - Hatayı düzeltmeye izin vermek
  - Programın otomatik olarak durmasına engel olmak