İleri Programlama

Hata Yakalama
(Exception Handling)

Hüseyin Ahmetoğlu

İstisna nedir?

- Java'da istisna, programın normal akışını bozan bir olaydır. Çalışma zamanında atılan bir nesnedir.
- İstisna İşleme, ClassNotFoundException, IOException, SQLException, RemoteException, vb. gibi çalışma zamanı hatalarını işlemek için kullanılan bir mekanizmadır.
- İstisna işlemenin temel avantajı , uygulamanın normal akışını sürdürmektir . Bir istisna uygulamanın normal akışını bozar, bu yüzden istisna işleme ile pragram çalışmaya devam eder.

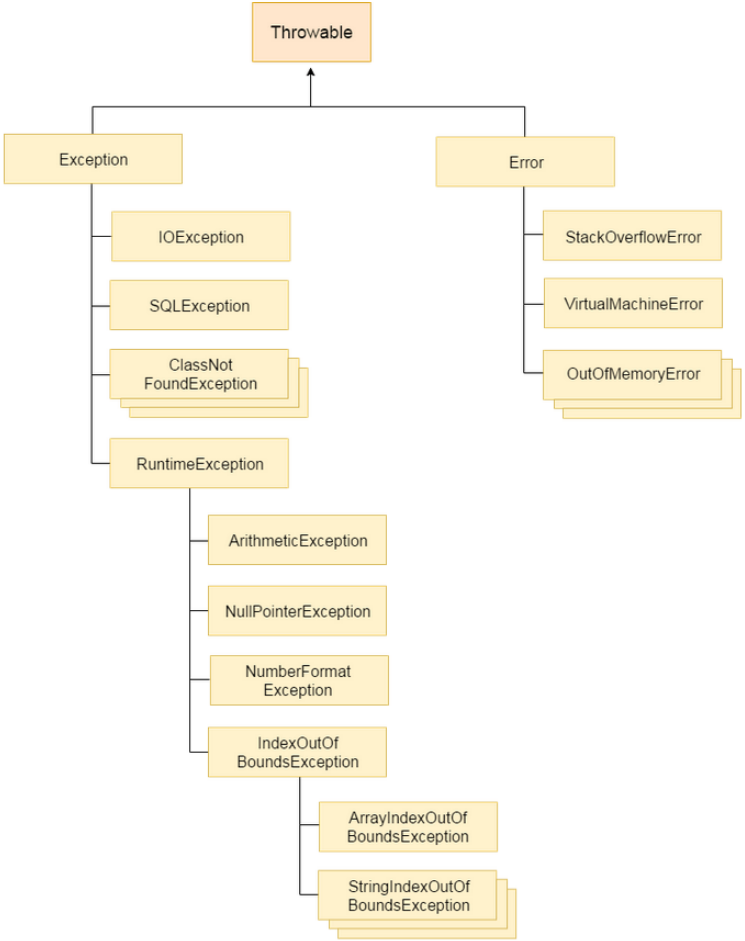
```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Java İstisna sınıflarının hiyerarşisi

Tüm **istisna** (*exception*) tipleri, yerleşik **Throwable** sınıfının altsınıflarıdır. **Throwable** sınıfı **Exception** ve **Error** olmak üzere iki alt sınıfa ayrılır.

Exception sınıfı da **RuntimeException** ve **IOException** başta olmak üzere bir çok alt sınıflara ayrılır. **RuntimeException** sınıfı içerisinde bulunan **istisna** (*exception*) tipleri yazılan program içerisinde otomatik olarak tanımlanır. **IOException** sınıfı içerisinde ise **giriş/çıkış** işlemleri ile ilgili istisnalar tanımlanır.

Error sınıfı ise normal şartlar altında programımız tarafından yakalanmayacak hataların sistem tarafından yakalanması için tanımlanan sınıftır. Error sınıfı hataları, normal koşullarda, program içinde yakalama imkanı yoktur. Çoğunlukla, programın çalışmasının durmasıyla sonuçlanır. Error tipindeki istisnalar, **Java sanal makinesinin** (*JVM*) kendisi ile ilgili hatalarını göstermek için Java sanal makinesi tarafından kullanılır.


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Kontrol Edilen ve Kontrol Edilmeyen İstisnalar Arasındaki Fark

- **1) Kontrol Edilen İstisna** RuntimeException ve Error dışında doğrudan Throwable sınıfını devralan sınıflar, kontrol edilen istisnalar olarak bilinir, örneğin IOException, SQLException vb. Kontrol edilen istisnalar derleme zamanında kontrol edilir.
- **2) Kontrol Edilmeyen İstisna** RuntimeException'ı devralan sınıflar denetlenmeyen istisnalar olarak bilinir, örneğin ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException vb. Kontrol edilmeyen istisnalar derleme zamanında kontrol edilmez, ancak çalışma zamanında kontrol edilir.
- **3) Hata** Hata düzeltilemez, örneğin OutOfMemoryError, VirtualMachineError, AssertionError vb.

Java İstisna Anahtar Kelimeleri

- **try**
  - "try" anahtar sözcüğü, istisna kodunu yerleştirmemiz gereken bir bloğu belirtmek için kullanılır. Try bloğunu ya catch ya da finally takip etmelidir. Bu, try bloğunu tek başına kullanamayacağımız anlamına gelir.
- **catch**
  - İstisnayı işlemek için "catch" bloğu kullanılır. Öncesinde try bloğu olmalıdır, bu da catch bloğunu tek başına kullanamayacağımız anlamına gelir. Daha sonra finally blok tarafından takip edilebilir.
- **finally**
  - "finally" bloğu, programın önemli kodunu yürütmek için kullanılır. Bir istisna işlenip işlenmediğine bakılmaksızın yürütülür.
- **throw**
  - Bir istisna atmak için "throw" anahtar sözcüğü kullanılır.
- **throws**
  - İstisnaları bildirmek için "throws" anahtar sözcüğü kullanılır. Bir istisna atmaz. Metotta bir istisna oluşabileceğini belirtir. Her zaman metot imzası ile kullanılır.

Java İstisna İşleme Örneği

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Java İstisnalarının Genel Senaryoları

- 1) ArithmeticException

```
int a=50/0;//ArithmeticException
```

- 2) NullPointerException

```
String s=null;
System.out.println(s.length());//NullPointerException
```

- 3) NumberFormatException

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

- 4) ArrayIndexOutOfBoundsException

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

try-catch

- Java try bloğu, bir istisna oluşturabilecek kodu içine almak için kullanılır.
- Try bloğunun belirli ifadesinde bir istisna oluşursa, blok kodunun geri kalanı yürütülmez. Bu nedenle, bir istisna atmayacak olan kodları try bloğunda tutmamanız önerilir.
- Java try bloğunu ya catch ya da finally bloğu takip etmelidir.

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

```
try{
//code that may throw an exception
}finally{}
```

Java catch block

- Java catch bloğu, parametre içindeki istisna türünü bildirerek İstisnayı işlemek için kullanılır.
- Bildirilen istisna, ana sınıf istisnası veya oluşturulan istisna türü olmalıdır.

```
public class TryCatchExample1 {

   public static void main(String[] args) {

      int data=50/0; //may throw exception

      System.out.println("rest of the code");

   }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Örnek 1

```
public class TryCatchExample2 {

   public static void main(String[] args) {
      try
      {
      int data=50/0; //may throw exception
      }
         //handling the exception
      catch(ArithmeticException e)
      {
         System.out.println(e);
      }
      System.out.println("rest of the code");
   }

}
```

```
java.lang.ArithmeticException: / by zero
rest of the code
```

Örnek 2

```
public class TryCatchExample3 {


    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
                        // if exception occurs, the remaining statement will not exceute
        System.out.println("rest of the code");
        }
            // handling the exception
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }


    }


}
```

```
java.lang.ArithmeticException: / by zero
```

Örnek 3

```
public class TryCatchExample4 {


   public static void main(String[] args) {
      try
      {
      int data=50/0; //may throw exception
      }
         // handling the exception by using Exception class
      catch(Exception e)
      {
         System.out.println(e);
      }
      System.out.println("rest of the code");
   }


}
```

```
java.lang.ArithmeticException: / by zero
rest of the code
```

Örnek 4

```
public class TryCatchExample5 {


    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
            // handling the exception
        catch(Exception e)
        {
                // displaying the custom message
            System.out.println("Can't divided by zero");
        }
    }


}
```

```
Can't divided by zero
```

Örnek 5

```
public class TryCatchExample6 {


   public static void main(String[] args) {
      int i=50;
      int j=0;
      int data;
      try
      {
      data=i/j; //may throw exception
      }
         // handling the exception
      catch(Exception e)
      {
          // resolving the exception in catch block
         System.out.println(i/(j+2));
      }
   }
}
```

```
25
```

Örnek 6

```
public class TryCatchExample7 {

   public static void main(String[] args) {

      try
      {
      int data1=50/0; //may throw exception

      }
          // handling the exception
      catch(Exception e)
      {
         // generating the exception in catch block
      int data2=50/0; //may throw exception

      }
   System.out.println("rest of the code");
   }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Örnek 7

```
public class TryCatchExample8 {


    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception


        }
            // try to handle the ArithmeticException using ArrayIndexOutOfBoundsException
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }


}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Örnek 8

```
public class TryCatchExample9 {


   public static void main(String[] args) {
      try
      {
      int arr[]= {1,3,5,7};
      System.out.println(arr[10]); //may throw exception
      }
         // handling the array exception
      catch(ArrayIndexOutOfBoundsException e)
      {
         System.out.println(e);
      }
      System.out.println("rest of the code");
   }


}
```

```
java.lang.ArrayIndexOutOfBoundsException: 10
rest of the code
```

Java Multi-catch block

- Bir try bloğunu bir veya daha fazla yakalama bloğu takip edebilir. Her bir yakalama bloğu farklı bir özel durum işleyicisi içermelidir. Bu nedenle, farklı istisnaların ortaya çıkması durumunda farklı görevler gerçekleştirmeniz gerekiyorsa, Java çoklu yakalama bloğunu kullanın.
- Bir seferde yalnızca bir istisna oluşur ve aynı anda yalnızca bir catch bloğu yürütülür.
- Tüm yakalama blokları en özelden en genele doğru sıralanmalıdır, yani ArithmeticException için yakalama, Exception için yakalamadan önce gelmelidir.

```
public class MultipleCatchBlock1 {

   public static void main(String[] args) {


        try{
            int a[]=new int[5];
            a[5]=30/0;
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
   }
}
```

```
Arithmetic Exception occurs
rest of the code
```

```java
public class MultipleCatchBlock2 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];

            System.out.println(a[10]);
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
    }
}
```

```
ArrayIndexOutOfBounds Exception occurs
rest of the code
```

```java
public class MultipleCatchBlock3 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
            System.out.println(a[10]);
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
    }
}
```

```
Arithmetic Exception occurs
rest of the code
```

```
public class MultipleCatchBlock4 {

   public static void main(String[] args) {

        try{
            String s=null;
            System.out.println(s.length());
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
   }
}
```

```
Parent Exception occurs
rest of the code
```

```
class MultipleCatchBlock5{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(Exception e){System.out.println("common task completed");}
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   System.out.println("rest of the code...");
 }
}
```

```
Compile-time error
```

İç içe try block

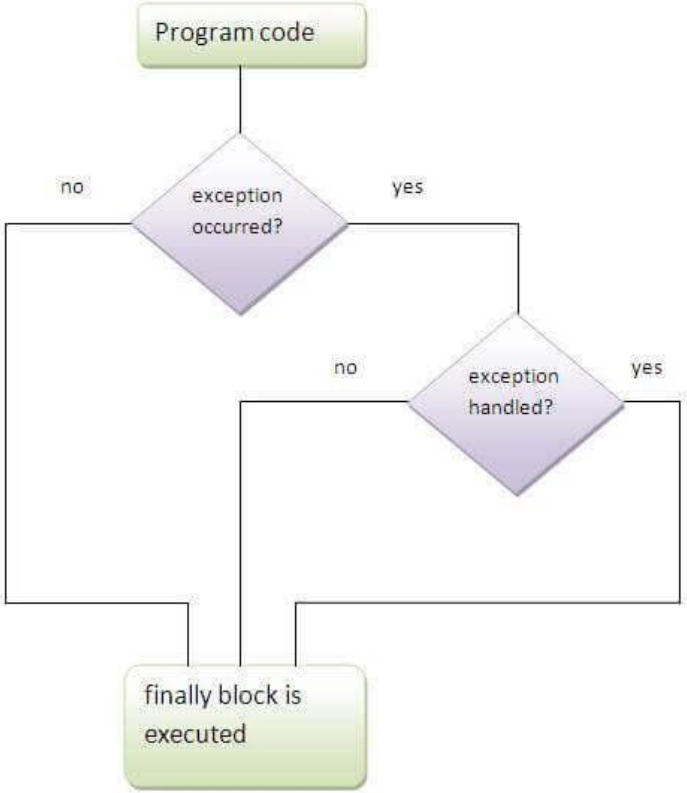
- Bazen bir bloğun bir bölümünün bir hataya neden olabileceği ve bloğun tamamının başka bir hataya neden olabileceği bir durum ortaya çıkabilir. Bu gibi durumlarda, istisna işleyicileri iç içe olmalıdır.

```
....
try
{
   statement 1;
   statement 2;
   try
   {
      statement 1;
      statement 2;
   }
   catch(Exception e)
   {
   }
}
catch(Exception e)
{
}
....
```

```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");
 }
}
```

Java finally block

- Java finally bloğu, bağlantıyı kapatma, akış vb. Gibi önemli kodları yürütmek için kullanılan bir bloktur.
- Java finally bloğu, istisna işlenip işlenmediğine bakılmaksızın her zaman yürütülür.
- Kural: Her bir try bloğu için sıfır veya daha fazla catch bloğu olabilir, ancak yalnızca bir tane finally bloğu olabilir.



Örnek

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:5
       finally block is always executed
       rest of the code...
```

Örnek

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:finally block is always executed
       Exception in thread main java.lang.ArithmeticException:/ by zero
```

Örnek

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       finally block is always executed
       rest of the code...
```

Java throw exception

- Java throw anahtar sözcüğü, açıkça bir istisna atmak için kullanılır.
- Java'da throw anahtar sözcüğü ile kontrol edilen veya kontrol edilmeyen istisna atabiliriz.
- throw anahtar sözcüğü esas olarak özel istisna atmak için metotolarla birlikte kullanılır.

```
public class TestThrow1{
  static void validate(int age){
   if(age<18)
    throw new ArithmeticException("not valid");
   else
    System.out.println("welcome to vote");
  }
  public static void main(String args[]){
    validate(13);
    System.out.println("rest of the code...");
  }
}
```

```
Exception in thread main java.lang.ArithmeticException:not valid
```

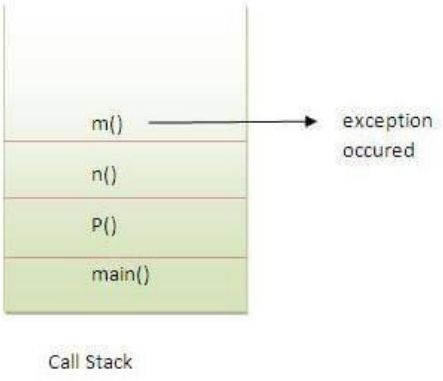
Java Exception propagation

- İlk önce yığının tepesinden bir istisna atılır ve yakalanmazsa, çağrı yığınını önceki yönteme bırakır.
- Kural: Varsayılan olarak Denetlenmeyen İstisnalar, çağrı zincirinde iletilir (yayılır).

```
class TestExceptionPropagation1{
  void m(){
    int data=50/0;
  }
  void n(){
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   TestExceptionPropagation1 obj=new TestExceptionPropagation1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Java Exception propagation

- Kural: Varsayılan olarak, Kontrol Edilen İstisnalar çağrı zincirinde iletilmez.




```
Output:Compile Time Error
```

```
class TestExceptionPropagation2{
 void m(){
   throw new java.io.IOException("device error");//checked exception
 }
 void n(){
   m();
 }
 void p(){
  try{
   n();
  }catch(Exception e){System.out.println("exception handeled");}
 }
 public static void main(String args[]){
  TestExceptionPropagation2 obj=new TestExceptionPropagation2();
  obj.p();
  System.out.println("normal flow");
 }
}
```

Java throws

- Java throws anahtar sözcüğü bir istisna bildirmek için kullanılır. Programcıya bir istisna olabileceğine dair bir bilgi verir, bu nedenle programcının istisna işleme kodunu sağlaması normal akışın sürdürülebilmesi için daha iyidir.
- İstisna İşleme, çoğunlukla kontrol edilen istisnaları işlemek için kullanılır. NullPointerException gibi kontrol edilemeyen bir istisna oluşursa, bu kod kullanılmadan önce kontrolün gerçekleşmediği anlamına gelir bu bir yazılımcı hatasıdır.
- Kural: Bir istisna bildiren bir yöntemi çağırıyorsanız, istisnayı yakalamanız veya bildirmeniz gerekir.

```
return_type method_name() throws exception_class_name{
//method code
}
```

```java
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

```
exception handled
normal flow...
```

Durum 1: İstisnayı işleme

İstisnayı ele almanız durumunda, program sırasında istisna olup olmadığına bakılmaksızın kod iyi yürütülecektir.

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}
public class Testthrows2{
   public static void main(String args[]){
    try{
     M m=new M();
     m.method();
    }catch(Exception e){System.out.println("exception handled");}

    System.out.println("normal flow...");
  }
}
```

```
Output:exception handled
       normal flow...
```

Durum 2: İstisnayı Bildirme

A) İstisnayı ilan etmeniz durumunda, istisna oluşmazsa, kod iyi yürütülür.

```
import java.io.*;
class M{
 void method()throws IOException{
  System.out.println("device operation performed");
 }
}
class Testthrows3{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();

    System.out.println("normal flow...");
  }
}
```

```
Output:device operation performed
       normal flow...
```

Durum 2: İstisnayı Bildirme

B) İstisna meydana gelirse istisna beyan etmeniz durumunda, fırlatmalar istisnayı işlemediğinden çalışma zamanında bir istisna atılır.

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}
class Testthrows4{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();

    System.out.println("normal flow...");
  }
}
```

```
Output:Runtime Exception
```

KAYNAKLAR

- Programming Notes. (2021, March 11). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming/index.html
- Tutorials - Javatpoint. (2021, June 06). Retrieved from https://www.javatpoint.com
- Çobanoğlu B. (2020) Java ile Programlama ve Veri Yapıları. İstanbul Pusula Yayıncılık. 978-605-2359-84-6