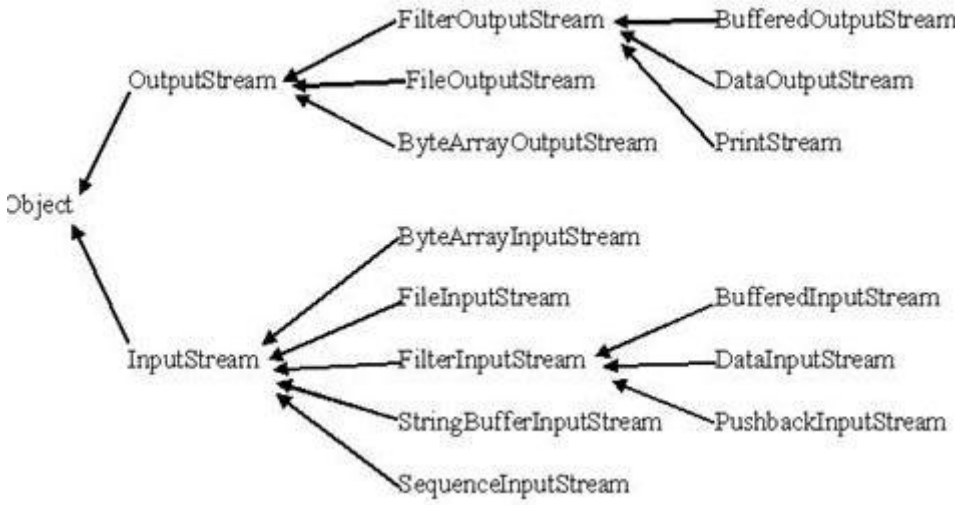
PART 15

15.1 Dosyaları Okuma ve Yazma

Stream, bir veri dizisi olarak tanımlanabilir. InputStream, bir kaynak dosyadan veri okumak için ve OutputStream bir hedef dosyaya veri yazmak için kullanılır.

Burada, Input ve Output stream'ler ile ilgili sınıfların hiyerarşisi gösterilmektedir.



Bu bölümde, iki önemli stream olan **FileInputStream** ve **FileOutputStream** ele alınacaktır.

15.2 FileInputStream

Bu stream, dosyalardan veri okumak için kullanılır. Nesneler, new anahtar sözcüğü kullanılarak oluşturulur ve birçok türde constructor mevcuttur.

Aşağıdaki constructor, dosya okuma amacıyla bir input stream nesnesi oluşturmak için türü string olan bir dosya ismi alır.

```
InputStream f = new FileInputStream("C:/java/hello");
```

Aşağıdaki constructor, dosya okuma amacıyla bir input stream nesnesi oluşturmak için bir dosya nesnesi alır. İlk olarak, File() metodunu kullanarak aşağıdaki gibi bir dosya nesnesi oluştururuz:

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Bir kere , *InputStream* nesnesi elinizde mevcutsa, stream okumak için kullanılabileceğiniz veya stream üzerinden farklı işlemler yapabileceğiniz listede gösterilen yardımcı metotları kullanabilirsiniz.

Methods with Description
public void close() throws IOException{} Bu metot , file output stream'i kapatır. Dosya ile ilgili sistem kaynaklarını bırakır. IOException atar.
public int read(byte[] r) throws IOException{} Bu metot, bir input stream'den dizinin içine r.length bit okur. Okunan bitlerin toplam sayısını döndürür. Eğer dosya sonu -1 ise döndürülecektir.
public int available() throws IOException{} Bir file input stream'den okunabilen bit sayısını verir. int türünde değişken döndürür.

15.3 FileOutputStream

FileOutputStream, bir dosya oluşturup içine veri yazmak için kullanılır. Stream, output için açılmadan önce, bir dosya yaratacaktır.(eğer önceden oluşturulmadıysa)

Burada, FileOutputStream nesnesi oluşturmak için kullanılan iki constructor gösterilmektedir.

Aşağıdaki constructor, dosyaya yazma amacıyla bir input stream nesnesi oluşturmak için türü string olan bir dosya ismi alır.

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Aşağıdaki constructor, dosyaya yazma amacıyla bir input stream nesnesi oluşturmak için bir dosya nesnesi alır. İlk olarak, File() metodunu kullanarak aşağıdaki gibi bir dosya nesnesi oluştururuz:

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Bir kere , *OutputStream* nesnesi elinizde mevcutsa, stream'e yazmak için kullanılabileceğiniz veya stream üzerinden farklı işlemler yapabileceğiniz listede gösterilen yardımcı metotları kullanabilirsiniz.

Methods with Description
public void close() throws IOException{} Bu metot, file output stream'i kapatır. Dosya ile ilgili sistem kaynaklarını bırakır. IOException atar
public void write() throws IOException{} Bu metot, output stream'e belirlenen bitte veri yazar.

Örnek:

Aşağıda, InputStream ve OutputStream örnekleri gösterilmektedir:

```
import java.io.*;

public class fileStreamTest{

   public static void main(String args[]){

   try{
      char[] data = {'a','b','c','d','e'};
      OutputStream outfile = new FileOutputStream("C:/test.txt");
      for(int x=0; x < data.length ; x++){
         outfile.write( data[x] ); // writes the bytes
      }
      outfile.close();

      InputStream infile = new FileInputStream("C:/test.txt");
      int size = infile.available();

      for(int i=0; i< size; i++){
         System.out.print((char) infile.read() + "  ");
      }
      infile.close();

   }catch(IOException e){
      System.out.print("Exception");
   }
   }
}
```

Yukarıdaki kod, test.txt dosyasını oluşturup, verilen karakterleri içine yazacaktır. Aynısı, stdout ekranında gösterilecektir.

15.4 Dosya Navigasyonu ve I/O

Dosya Navigasyonu ve I/O temellerini öğrenmek için inceleyeceğimiz birkaç sınıf vardır.

- FileReader Sınıfı
- FileWriter Sınıfı
- File Sınıfı

File Reader

Bu sınıf, InputStream sınıfından miras alır. FileReader, karakter streamlerini okuma için kullanılır.

Aşağıdaki syntax , okumak için File nesnesi verilen yeni bir FileReader nesnesi oluşturur.

```
FileReader(File file)
```

File Writer

Bu sınıf, OutputStream sınıfından miras alır. Bu sınıf, karakter streamlerini yazma için kullanılır.

Aşağıdaki syntax, File nesnesi verilen bir FileWriter nesnesi oluşturur.

```
FileWriter(File file)
```

Örnek:

Aşağıda bu sınıflarla ilgili örnek gösterilmektedir:

```
import java.io.*;

public class FileRead{

   public static void main(String args[])throws IOException{

      File file = new File("Hello1.txt");
      // creates the file
      file.createNewFile();
      // creates a FileWriter Object
      FileWriter writer = new FileWriter(file);
```

```
      // Writes the content to the file
      writer.write("This\n is\n an\n example\n");
      writer.flush();
      writer.close();

      //Creates a FileReader Object
      FileReader fr = new FileReader(file);
      //Creates a LineNumberReader Object
      LineNumberReader lnreader = new LineNumberReader(fr);
      String line = "";
      while ((line = lnreader.readLine()) != null) {
            System.out.println(lnreader.getLineNumber() + ": " + line);
      }
      fr.close();
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
1: This
2:  is
3:  an
4:  example
```

15.4 Java Dizinleri

Dizinleri Oluşturma

Burada, dizin oluşturmak için kullanılan iki adet File yardımcı metotları gösterilmektedir:

- **mkdir()** metod dizin oluşturur, başarılı(succes) durumda true değer, başarısız(failure) durumda iste false değer döndürür. Failure, File nesnesinde belirlenen yolun zaten var olması veya bütün yolun var olmamasından dolayı dizinin oluşturalamamasını belirtir.
- **mkdirs()** metodu, hem dizini oluşturur hem de dizinin bütün üst öğelerini oluşturur.

Aşağıdaki örnek, “c:/java/example/folder” dizinini oluşturur:

```
import java.io.File;

public class CreateDir {
   public static void main(String args[]) {
      String dirname = "c:/java/example/folder";
      File d = new File(dirname);
      // Create directory now.
      if(d.mkdirs())
       { System.out.print(“created”); }
```

```
      else
       { System.out.print("error! Not created"); }
  }
}
```

Not: Java, UNIX ve Windows'taki yol ayırıcılarının kurallara göre olmasına otomatik olarak dikkat eder. Eğer , Java'nın Windows versiyonunda ileri slash(/) kullanırsanız, yol yinede doğru şekilde çözümlenecektir.

Dizinleri okuma

Dizin, diğer dosyaların ve dizinlerin listesini içeren bir File nesnesidir. Bir File nesnesi yarattığınızda ve bu bir dizinse, isDirectory() metodu true değer döndürecektir.

Nesnenin içindeki diğer dosyalar ve dizinlerin listesini çıkartmak için list() metodunu çağırabilirsiniz. Burada gösterilen program, dizin içeriğini almak için list() metodunun ne şekilde kullanacağınızı örneklemektedir.

```
import java.io.File;

public class DirList {
   public static void main(String args[]) {
      String dirname = "c:/mysql";
      File f1 = new File(dirname);
      if (f1.isDirectory()) {
         System.out.println( "Directory of " + dirname);
         String s[] = f1.list();
         for (int i=0; i < s.length; i++) {
            File f = new File(dirname + "/" + s[i]);
            if (f.isDirectory()) {
               System.out.println(s[i] + " is a directory");
            } else {
               System.out.println(s[i] + " is a file");
            }
         }
      } else {
         System.out.println(dirname + " is not a directory");
    }
  }
}
```

Bu aşağıdaki sonucu üretecektir:

```
Directory of /mysql
bin is a directory
lib is a directory
demo is a directory
test.txt is a file
README is a file
index.html is a file
include is a directory
```

PART 16

16. İstisnalar(Exceptions)

İstisna, programın çalışması sırasında ortaya çıkan bir sorundur. Bir istisna ,birçok değişik nedenden dolayı meydana gelebilir, aşağıdaki gibi:

- Kullanıcı geçersiz bir veri girebilir.
- Açılması geren dosya bulunamayabilir.
- İletişimin ortasından ağ bağlantısı kaybolabilir veya JVM belleği tükenebilir.

Bu istisnaların bazılarına kullanıcı hatası sebep olur, diğerleri programcının hatası ve fiziksel kaynaklardan dolayı olabilir.

İstisna işlemenin Javada nasıl çalıştığını anlamak için, istisnaların üç kategorisini anlamanız geremektedir.

- **Kontrol edilmiş istisnalar(Checked exceptions)**: Kontrol edilmiş istisnalar, genellikle kullanıcı hatları veya programcı tarafından öngörülememiş sorunlardır. Örnek olarak, bir dosya açılıcak ancak dosya bulanamıyorsa, istisna meydana gelir.
- **Çalışma anındaki istisnalar(Runtime exceptions)**: Çalışma anındaki istisnalar, muhtemelen programcı tarafından önlenebilir istisnalardır.
- **Hatalar**: Başlı başına istisna değildir, kullanıcı veya programcının kontrolü dışında ortaya çıkan sorunlardır. Örnek, bir stack overflow meydana gelirse, bir hata ortaya çıkacaktır. Bunlar, derleme süresinde yok sayılır.

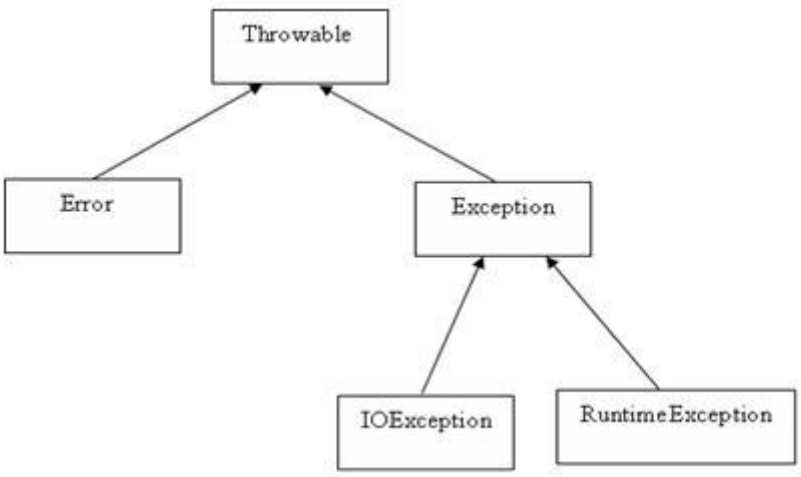
16.1 İstisna Hiyerarşisi

Bütün istisna sınıfları, java.lang.Exception sınıfının alt türleridir. Exception sınıfı, Throwable sınıfının alt sınıfıdır. Throwable sınıfından elde edilen diğer bir alt sınıfta Error sınıfıdır.

Hatalar, normalde Java programları tarafından tutulmazlar. Bu koşullar normalde java programları tarafından işlenmeyen ciddi başarısızlık durumunda meydana gelirler. Hatalar, çalıştırma ortamı tarafından oluşturulan hataları belirtmek için oluşturulur.Örnek: JVM belleği tükenmiş. Normalde programlar hatalardan kurtulamaz.

Exception sınıfı iki ana altsınıfa sahiptir:

- IOException Sınıfı
- RuntimeException Sınıfı



16.2 Java Dahili İstisnaları

En genel istisnalar, standard tip RuntimeException sınıfının altsınıflarıdır.

Java, çeşitli sınıf kütüphaneleri ile ilgili bir çok türde istisna tanımlar. Aşağıdaki liste, Unchecked RuntimeException'ları göstermektedir.

Exception	Description
ArithmeticException	Aritmetic hata, sıfıra bölünme gibi
ArrayIndexOutOfBoundsException	Dizi index'i sınırların dışında.
ArrayStoreException	Dizi elemanına uyumsuz türde atama.
ClassCastException	Geçersiz cast.
IllegalArgumentException	Metodu çalıştırmak için illegal argüman.
IllegalMonitorStateException	Illegal monitor işlemi, unlocked bir thread üzerinde bekleme gibi.
IllegalStateException	Ortam veya uygulama doğru olmayan durumda.
IllegalThreadStateException	İstenen işlem anlık thread durumu ile uyumlu değil.
IndexOutOfBoundsException	Bazı türde index sınır dışında.
NegativeArraySizeException	Dizi negatif boyutta yaratılmış.
NullPointerException	Null referansının geçersiz kullanımı.
NumberFormatException	Bir string'in sayısal formata geçersiz dönüşümü.
SecurityException	Güvenlik ihlali denemesi.
StringIndexOutOfBounds	Bir stringe sınırları dışında index verme.
UnsupportedOperationException	Desteklenmeyen işlem karşılatırıldı.

Aşağıdaki liste, Javada ki Checked Exception'ları göstermektedir:

Exception	Description
ClassNotFoundException	Sınıf bulunamadı.
CloneNotSupportedException	Nesne kopyalama, Kopyalanabilir arayüzde uygulanamadı.
IllegalAccessException	Sınıfa erişim engellendi.
InstantiationException	Abstract bir sınıf veya arayüzde nesne oluşturma.
InterruptedException	Bir thread, diğer bir thread tarafından kesildi.
NoSuchFieldException	İstenen dosya yok.
NoSuchMethodException	İstenen metot yok.

16.3 İstinaları Yakalamak

Bir metot, try ve catch anahtar kelimelerinin kombinasyonunu kullanarak bir istisnayı yakalayabilir. Try/catch bloğu, istisna oluşması muhtemel kodun etrafına yerleştirilir. Try/catch bloğu içindeki kod, korumalı kod(protected code) olarak adlandırılır ve try/catch kullanımı aşağıdaki gibidir:

```
try
{
   //Protected code
}catch(ExceptionName e1)
{
   //Catch block
}
```

Bir catch ifadesi, yakalamak istediğiniz istisnanın deklare edilmiş türünü içerir. İstisna korumalı kod içinde oluşuyorsa, catch bloğu(veya blokları), try bloğunu kontrol eder. Eğer istisna türü catch bloğu içinde listelenmiş istisnalardan biriyse, istisna catch bloğuna geçirilir.

Örnek

Aşağıdaki dizi, 2 elemanlı bir dizi olarak deklare edilmiştir. Ardından kod, 3. elemete erişmeyi deniyor.Burada bir istisna oluşmaktadır:

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{

   public static void main(String args[]){
      try{
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }

   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
```

16.4 Birden çok catch Bloğu

Bir try bloğu, birden çok catch bloğu tarafından takip edilebilir. Birden çok catch bloğunun syntax'ı aşağıdaki gibidir:

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}
```

Önceki ifadeler üç catch bloğu göstermektedir, ama siz tek bir try bloğundan sonra herhangi bir sayıda catch bloğu kullanabilirsiniz. Eğer istisna korumalı kod içinde meydana gelirse, listedeki ilk catch bloğuna atılır. Atılan istisnanın veri tipi ExceptionType1 ile eşleşiyorsa, burada yakalanır. Eşleşmiyorsa, istisna ikince catch bloğuna geçer. Bu işlem istisna yakalanana kadar devam eder, bu durumda metot çalışmayı durdurur ve istisna call stack üzerindeki bir önceki metodu altına alır.

Örnek

Birden çok try/catch ifadesinin kullanımını gösteren kod parçası:

```
try
{
   file = new FileInputStream(fileName);
   x = (byte) file.read();
}catch(IOException i)
{
   i.printStackTrace();
   return -1;
}catch(FileNotFoundException f) //Not valid!
{
   f.printStackTrace();
   return -1;
}
```

16.5 throws/throw Anahtar Kelimeleri

Eğer bir metot kontrol edilmiş bir istisnayı işleyemiyorsa, o zaman **throws** anahtar kelimesi kullanarak deklare etmek zorundadır. throws anahtar kelimesi metot

imzasının sonunda görülür. **Throws** anahtar kelimesi istisnayı bir başka sınıfa gönderir.

Aşağıdaki metot bir RemoteException atmanın deklare edilmesidir:

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

Bir metot, birden çok istisnanın atılmasını deklare eder, bu durumda istisnalar virgülle ayrılarak bir listede deklare edilir. Örnek, aşağıdaki metot bir RemoteException ve bir InsufficientFundsException atılmasını deklare ediyor:

```
import java.io.*;
public class className
{
public void withdraw(double amount)throws RemoteException, InsufficientFundsException
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

16.6 finally Anahtar Kelimesi

Finally anahtar kelimesi, try bloğunun ardından gelen bir kod bloğu oluşturmak için kullanılır. Bir finally kod bloğu, istisna olsa da olmasa da, her zaman çalışır.

Catch bloklarından sonra gelen finally bloğunun syntax'I şu şekildedir.

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
```

```
}finally
{
   //The finally block always executes.
}
```

Örnek

```
public class ExcepTest{

   public static void main(String args[]){
      int a[] = new int[2];
      try{
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      finally{
         a[0] = 6;
         System.out.println("First element value: " +a[0]);
         System.out.println("The finally statement is executed");
      }
   }
}
```

Bu aşağıdaki sonucu üretecektir:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

Aşağıdakilere dikkat edin:

- Bir catch koşulu, try ifadesi olmadan var olamaz.
- try/catch bloğu olsa bile, finally koşulu kullanmak zorunlu değildir.
- Catch koşulu veya finally koşulu olmazsa, try bloğu olmaz.