Εξαιρέσεις (Exceptions)

Γιώργος Θάνος
gthanos@uth.gr
Γραφείο: Γ5/8, 3ος όροφος
Γκλαβάνη 37

Εισαγωγικά

Κατά την εκτέλεση ενός προγράμματος, όταν συμβαίνει κάποιο αναπάντεχο ή μη διαχειρίσιμο από το πρόγραμμα γεγονός συμβαίνουν τα εξής:

1. δημιουργείται ένα νέο αντικείμενο το οποίο ονομάζεται εξαίρεση (exception).
2. η ροή εκτέλεσης του προγράμματος διακόπτεται.

Για να διασφαλίσουμε ότι το πρόγραμμά μας θα συνεχίσει την εκτέλεσή του ακόμη και αν συμβεί ένα απροσδόκητο γεγονός, χρησιμοποιούμε τον μηχανισμό διαχείρισης των εξαιρέσεων.

Το αντικείμενο της εξαίρεσης

```
import java.util.Scanner;
public class TestDivideByZero {
   public static void main (String[] args) {
      int x, y;
      Scanner input = new Scanner(System.in);
      System.out.print( "Enter first integer: " );
      x = input.nextInt();
      System.out.print( "Enter second integer: " );
      y = input.nextInt();
      System.out.printf( "Product is %d\n", x/y );
   }
}
```

Τι θα συμβεί στο διπλανό πρόγραμμα αν ο 2ος αριθμός που θα δώσει ο χρήστης είναι 0;

Επίσης αν ο χρήστης δεν δώσει αριθμό σαν είσοδο αλλά γράμμα;

Καλούμαστε να προβλέψουμε όλες τις περιπτώσεις στις οποίες ο χρήστης είτε από λάθος είτε επειδή είναι κακόβουλος θα δώσει σαν είσοδο κάτι το οποίο δεν είναι σωστό και θα προκαλέσει πρόβλημα.

Το αντικείμενο της εξαίρεσης

Στο προηγούμενο παράδειγμα εφόσον ο χρήστης εισάγει 0, το σύστημα θα παράξει ένα αντικείμενο που θα περιγράφει το είδος του λάθους που συνέβη. Ο προγραμματιστής πρέπει:

1. να λάβει το αντικείμενο της εξαίρεσης
2. να εξετάσει το είδος του λάθους που έχει προκύψει.
3. να αντιμετωπίσει το λάθος, ώστε το πρόγραμμά να επιστρέψει σε ένα σημείο από όπου μπορεί να συνεχίσει τη λειτουργία του χωρίς πρόβλημα.

Διαχείριση Εξαιρέσεων - Try-Catch Block

```
try {
   /*  code that may throw an exception here. */
} catch (ExceptionTypeOne ex) {
   /* exception handler for ExceptionTypeOne objects. */
} catch (ExceptionTypeTwo ex) {
   /* exception handler for ExceptionTypeTwo objects. */
}
```

```
import java.util.*;
public class DivideByZero {
  public static void main (String[] args) {
    int x, y=1; Scanner input = new Scanner(System.in);
    try {
      System.out.print( "Enter first integer: " );
      x = input.nextInt();
      System.out.print( "Enter second integer: " );
      y = input.nextInt();
      System.out.printf( "Product is %d\n", x/y );
    } catch (ArithmeticException ae) {
      System.out.println("ArithmeticException occured!");
      if(y == 0){ System.out.println("Division by zero in particular"); }
    }
    catch(InputMismatchException ex) {
      System.out.println("No integer value was specified!");
    }
  }
}
```

1ο Παράδειγμα

Η κλάση `TestDivideByZero` μπορεί να γραφεί ως εξής ώστε να διαχειρίζεται τις πιθανές εξαιρέσεις που θα παραχθούν.

2ο Παράδειγμα

```
import java.io.*;
public class WholeFileReader {
  public static String readFile(String path) {
    try {
      File file = new File (path);
      FileReader fReader = new FileReader(file);
      BufferedReader in = new BufferedReader(fReader);
      String inputLine;
      StringBuffer strDocument = new StringBuffer();
      while ((inputLine = in.readLine()) != null) {
        strDocument.append(inputLine);
            //if(inputLine.length() == 0 )
        //  throw new IOException();
      }
      fReader.close();
      return strDocument.toString();
    }
    catch(IOException ex) {
      System.out.println("IOException occured " +
                    "while reading from file "+path);
    }
    catch(FileNotFoundException ex) {
      System.out.println("The specified file " +
                          "was not found at "+ path);
      return "";
    }
    return "Nothing to return..";
  }
  public static void main(String args[]) {
    try { System.out.println(
          WholeFileReader.readFile(args[0]) );
    } catch(IndexOutOfBoundsException ex) {
      System.out.println("No file specified " +
"from command line!\n");
    }
  }
```

Διάβασμα των περιεχομένων ενός αρχείου κειμένου μέσω της στατικής μεθόδου `readFile`.

Περισσότερα του ενός catch blocks - Ιεράρχιση της σειράς εμφάνισης τους

Παρατηρήστε ότι εμφανίζονται δύο **catch** blocks.

Το πρώτο catch block αφορά **`IOException`** objects, ενώ το δεύτερο catch block αφορά **`FileNotFoundException`** objects.

Από τους συνδέσμους που παρατίθενται θα παρατηρήσετε ότι το **`FileNotFoundException`** είναι υποκλάση του **`IOException`** και δηλώνει ότι το αρχείο δεν βρέθηκε στο filesystem.

Ερωτήσεις:

1. Τι θα γινόταν εάν αφαιρούσαμε το **`FileNotFoundException`** catch block;
2. Τι θα γινόταν εάν αντιστρέφαμε τη σειρά των **`FileNotFoundException`** και **`IOException`** catch blocks;

Finaly Block

Εκτός από τα **`catch blocks`** τα οποία εκτελούνται όταν έχουμε κάποιο exception, μπορούμε να προσθέσουμε ένα **`finaly`** block το οποίο θα εκτελεστεί σε κάθε περίπτωση.

Το **`finaly`** block θα εκτελεστεί στις παρακάτω περιπτώσεις:

1. Εάν προκύψει η εξαίρεση που έχουμε φροντίσει να διαχειριστούμε.
2. Εάν προκύψει μια εξαίρεση ενός τύπου που δεν έχουμε φροντίσει να διαχειριστούμε.
3. Εάν δεν προκύψει καμία απολύτως εξαίρεση.

Παράδειγμα finally block

```
import java.io.*;
public class WholeFileReader {
  public static String readFile(String path) {
    try {
      File file = new File (path);
      FileReader fReader = new FileReader(file);
      BufferedReader in = new BufferedReader(fReader);
      String inputLine;
      StringBuffer strDocument = new StringBuffer();
      while ((inputLine = in.readLine()) != null) {
        strDocument.append(inputLine);
            //if(inputLine.length() == 0 )
        //  throw new IOException();
      }
      fReader.close();
      return strDocument.toString();
    }
    catch(FileNotFoundException ex) {
      System.out.println("The specified file " +
                         "was not found at "+ path);
    }
    catch(IOException ex) {
      System.out.println("IOException occured " +
                   "while reading from file "+path);
    }
    finally {
      if(fReader!=null) {
        try{ fReader.close();
        } catch(IOException ex) {
          System.out.println("IOException occured "+
                   "while closing file.");
        } // end of catch
      }   // end of if
    }     // end of finally block
    return "";
  }
}
```

Finally Block

Ο λόγος που συνήθως χρησιμοποιήσουμε το **`finally block`** είναι για να συμπεριλάβουμε κώδικα που θέλουμε να εκτελεστεί σε όλες τις περιπτώσεις (π.χ. να κλείσουμε ελεγχόμενα τα αρχεία του προγράμματος ή να κλείσουμε δικτυακές συνδέσεις.

Στο προηγούμενο παράδειγμα παραλλάσσεται η μέθοδος **`readFile`** του αρχικού παραδείγματος, ώστε στο **`finally`** block η μέθοδος να κλείνει το αρχείο που άνοιξε.

Η διαφορά σε σχέση με το αρχικό παράδειγμα είναι ότι ακόμη και εάν δημιουργηθεί μία εξαίρεση την ώρα που διαβάζουμε, η ροή του προγράμματος θα περάσει από το **`finally`** block και το αρχείο τελικά θα κλείσει. Αυτό δεν ισχύει για το αρχικό παράδειγμα.

Χειρισμός της εξαίρεσης σε υψηλότερο επίπεδο

Ας υποθέσουμε ότι έχετε τις μεθόδους **main**, **method1**, **method2**, **method3** όπου η ιεραρχία κλήσεων μεταξύ τους είναι η εξής:

- Η **main** καλεί την **method1**.
- Η **method1** καλεί την **method2**.
- Η **method2** καλεί την **method3**.

Εάν συμβεί μία εξαίρεση μέσα στη μέθοδο **method3** τότε αυτή μπορούμε να τη διαχειριστούμε είτε μέσα στην **method3**, είτε μέσα στις **method2**, **method1** και **main**.

Αντίστοιχα, εάν συμβεί μία εξαίρεση μέσα στη μέθοδο **method2** τότε αυτή μπορούμε να τη διαχειριστούμε είτε μέσα στην **method2**, είτε μέσα στις **method1** και **main**.

Η **main** είναι η τελευταία μέθοδος στην οποία μπορεί να γίνει διαχείριση της εξαίρεσης. Εάν η διαχείριση δεν γίνει ούτε στη **main**, τότε το πρόγραμμα τερματίζει εκτυπώνοντας στην κονσόλα πληροφορίες για την εξαίρεση που δημιουργήθηκε.

```
import java.io.*; import java.lang.*;
public class WholeFileReader {
  public String readFile(String path) throws
 FileNotFoundException {
    FileReader fReader = null;
    try {
      File file = new File (path);
      fReader = new FileReader(file);
      BufferedReader in = new BufferedReader(fReader);
      String inputLine;
      StringBuffer strDocument = new StringBuffer();
      try {
        while ((inputLine = in.readLine()) != null) {
          strDocument.append(inputLine);
        }

      } // end of inner try
      catch(IOException ex) {
        System.out.println("IOException occured while"
                        +" reading from file "+path);
      }
      return strDocument.toString();
    } // end of outer try
```

Παράδειγμα διαχείρισης της εξαίρεσης σε υψηλότερο επίπεδο

```
    finally {
      if( fReader != null) {
        try {
          System.out.println("Closing file");
          fReader.close();
        }
        catch(IOException ex) {
          System.out.println("IOException occured while
closing file "+path);
        }
      }
    } // end of finally block
  }   // end of readFile method

  public static void main(String args[]) {
    WholeFileReader wfr = new WholeFileReader();
    try {
      System.out.println(wfr.readFile(args[0]) );
    }
    catch(IndexOutOfBoundsException ex) {
      System.out.println("No file has been
specified!\n");
    }
    catch(FileNotFoundException ex) {
      System.out.println("The specified file was not"
.                              + "found at "+ args[0]);
    }
  }

}
```

Δημιουργία νέων τύπων εξαιρέσεων

Μπορούμε να δημιουργήσουμε ένα νέο τύπο εξαίρεσης δημιουργώντας μία κλάση που κληρονομεί από την java.lang.Exception.

```
public class EmptyFileException extends java.lang.Exception {

}
```

Πυροδότηση εξαίρεσης

Όπως τονίσαμε και παραπάνω η εξαίρεση είναι και αυτή ένα αντικείμενο. Για να καλέσουμε ένα αντικείμενο εξαίρεσης το μόνο που έχουμε να κάνουμε είναι να βάλουμε τη δεσμευμένη λέξη **throw** και δίπλα το αντικείμενο της εξαίρεσης που θέλουμε να καλέσουμε.

Στο παράδειγμα, εάν μετά την ολοκλήρωση διαβάσματος του αρχείου αντιληφθούμε ότι το αρχείο που διαβάσαμε ήταν κενό τότε το πρόγραμμα πυροδοτεί το **`EmptyFileException`** (δες προηγούμενη διαφάνεια).

```
public class WholeFileReader {
  public String readFile(String path) throws
 FileNotFoundException, EmptyFileException {
    FileReader fReader = null;
    try {
      File file = new File (path);
      fReader = new FileReader(file);
      BufferedReader in = new BufferedReader(fReader);
      String inputLine;
      StringBuffer strDocument = new StringBuffer();
      try {
        while ((inputLine = in.readLine()) != null) {
          strDocument.append(inputLine);
        }
      }
      catch(IOException ex) {
        System.out.println("IOException occured while "+
.                          "reading from file "+path);
      }
      if( strDocument.toString().isEmpty() ) {
         throw new EmptyFileException();
      }
      return strDocument.toString();
    }

    finally {
      if( fReader != null) {
        try {
          fReader.close();
        }
        catch(IOException ex) {
         System.out.println("IOException when closing file");
        }
      }
    }
  }
```

```
  public static void main(String args[]) {
     WholeFileReader wfr = new WholeFileReader();
     try {
       System.out.println(wfr.readFile(args[0]) );
     }
     catch(IndexOutOfBoundsException ex) {
       System.out.println("No file has been"+
.                         " specified!");
     }
     catch(FileNotFoundException ex) {
       System.out.println("The specified file was"+
        .                  " not found at "+ args[0]);
     }
     catch(EmptyFileException ex) {
       System.out.println("File is empty!!!");
     }
   }

}
```

Δημιουργία εξαίρεσης από το πρόγραμμα, εφόσον το αρχείο είναι κενό.

try-with-resources block

```
import java.io.*;
public class WholeFileReader {
  public String readFile(String path) {
    try ( FileReader fReader = new FileReader(new File (path)) ) {
      BufferedReader in = new BufferedReader(fReader);
      StringBuffer strDocument = new StringBuffer();
      while ((String inputLine = in.readLine()) != null) {
        strDocument.append(inputLine);
      }
      return strDocument.toString();
    }
    catch(IOException ex) {
      System.out.println("IOException occured while reading from file "+path);
    }
    return "";
  }
}
```

Δεν χρειάζεται να ανησυχούμε αν θα κλείσει σωστά το αρχείο.

Το JVM θα αναλάβει να κλείσει το FileReader object αν αυτό παραμένει ανοιχτό.

Απαραίτητη προϋπόθεση για να συμβεί το παραπάνω είναι η κλάση του αντικειμένου που θα δημιουργηθεί να υποστηρίζει το interface **java.lang.Autoclosable** ή ένα από τα sub-interfaces αυτού (π. χ. java.io.Closable).

Κατηγορίες Εξαιρέσεων (⅓)

`Checked Exceptions:` Σε αυτή την κατηγορίες ανήκουν όλες οι εξαιρέσεις για τις οποίες ένα πρόγραμμα θα περιμέναμε να συνεχίσει κανονικά την λειτουργία του όταν συμβούν.

Για παράδειγμα, εάν προσπαθήσουμε να ανοίξουμε ένα αρχείο που δεν υπάρχει στον δίσκο τότε μία εξαίρεση του τύπου **`FileNotFoundException`** θα πρέπει να παραχθεί. Η εξαίρεση αυτή δεν σηματοδοτεί και το τέλος του προγράμματος, αντιθέτως ένα καλά γραμμένο πρόγραμμα θα πρέπει να ειδοποιήσει τον χρήστη του ότι δεν βρέθηκε το αρχείο προκειμένου να επιλέξει ένα άλλο αρχείο. Ο compiler της Java απαιτεί την διαχείριση των εξαιρέσεων αυτής της κατηγορίας. Οι εξαιρέσεις αυτή της κατηγορίας είναι απόγονοι της κλάσης **`java.lang.Exception`**.

Κατηγορίες Εξαιρέσεων (⅔)

`Errors:` Στην κατηγορία αυτή ανήκουν εξαιρέσεις που συμβαίνουν για λόγους που δεν εξαρτώνται από το πρόγραμμα. Για παράδειγμα αφού ανοίξετε ένα αρχείο για διάβασμα κάποιος τραβάει το flash drive από το οποίο διαβάζατε. Σε αυτή την περίπτωση παράγεται ένα **IOError** (και το πρόγραμμα σας είναι αδύνατον να συνεχίσει την λειτουργία του. Ο compiler της Java ΔΕΝ απαιτεί την διαχείριση αυτών των εξαιρέσεων. Οι εξαιρέσεις αυτή της κατηγορίας είναι απόγονοι της κλάσης java.lang.Error.

Κατηγορίες Εξαιρέσεων (3/3)

`Runtime Exceptions`: Σε αυτή την κατηγορία ανήκουν οι εξαιρέσεις που είναι κατά κανόνα αποτέλεσμα φτωχών προγραμματιστικών πρακτικών. Για παράδειγμα, εάν προσπαθώντας να ανοίξετε ένα αρχείο δώσετε για διάβασμα περάσετε στον κατασκευαστή της κλάσης **`FileReader`** την τιμή null, τότε θα παραχθεί μία εξαίρεση του τύπου **`NullPointerException`**. Μπορείτε να διαχειριστείτε τέτοιες εξαιρέσεις αν και μάλλον είναι προτιμότερο να προσπαθήσετε να εντοπίσετε το προγραμματιστικό λάθος. Ο *compiler* της Java ΔΕΝ απαιτεί την διαχείριση αυτών των εξαιρέσεων. Οι εξαιρέσεις αυτή της κατηγορίας είναι απόγονοι της κλάσης **`java.lang.RuntimeException`**.