ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ

Ενότητα 10: Exceptions handling (Χειρισμός εξαιρέσεων)

Μιχάλης Δρακόπουλος

Σχολή Θετικών επιστημών

Τμήμα Μαθηματικών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ (Java)
Ενότητα 10

Exceptions handling (Χειρισμός εξαιρέσεων)

Exception: σφάλμα που προκλήθηκε κατά την εκτέλεση του προγράμματος και που μπορεί να αντιμετωπιστεί από τον προγραμματιστή.

Error: σοβαρό σφάλμα που οφείλεται σε πρόβλημα που δεν μπορεί να αντιμετωπιστεί (π.χ., out of memory, stack overflow, κτλ) [θεωρητικά μπορεί να αντιμετωπιστεί, αλλά πρακτικά δεν αντιμετωπίζεται, αφού η λειτουργία του προγράμματος μετά από ένα τέτοιο σφάλμα είναι συνήθως προβληματική].

Τα exceptions μπορούν να αντιμετωπιστούν, άρα χρειάζονται ειδικό χειρισμό (exceptions handling).

Η κλάση `Exception` ορίζει απλές συνθήκες λάθους που μπορεί να συναντήσει κάποιο πρόγραμμα. Αντί να αφήνουμε το πρόγραμμα να τερματιστεί, μπορούμε να γράψουμε κώδικα που να χειρίζεται τις εξαιρέσεις και να συνεχίζει την εκτέλεση του προγράμματος. Όταν συμβεί κάποιο σφάλμα στο πρόγραμμα, ο κώδικας που το ανακαλύπτει «μεταβιβάζει» (`throws` – ο αγγλικός όρος) μια εξαίρεση.

→ Αν «συλλάβουμε» (`catch`) την εξαίρεση, είναι δυνατόν να επανακάμψουμε το πρόγραμμα ή να δώσουμε στο χρήστη κατάλληλα μηνύματα/οδηγίες. Η «σύλληψη» γίνεται εγκλωβίζοντας τον κώδικα που μπορεί να πετάξει/μεταβιβάσει μια εξαίρεση σε ένα `try`-block και τοποθετώντας στη συνέχεια ένα τουλάχιστον `catch`-block. Στο σημείο που θα συμβεί η εξαίρεση, η εκτέλεση του κώδικα διακόπτεται και η ροή του συνεχίζεται στο `catch`-block.

```
try
{
      // κώδικας που μπορεί να προκαλέσει μια συγκεκριμένη εξαίρεση
}
catch(ExceptionType e)
{
       // κώδικας που θα εκτελεστεί αν προκύψει η ExceptionType
}
```

Μπορεί να υπάρχουν και περισσότερα `catch` σε ένα `try`. Ο έλεγχος περνάει από το πρώτο στο επόμενο κτλ., μέχρι να βρεθεί το κατάλληλο. → Γι αυτό, ξεκινάμε από το πιο ειδικό (εξειδικευμένο) `catch` στο πιο γενικό.

Υπάρχει προαιρετικά και το block `finally` που περιέχει κώδικα ο οποίος θα εκτελεστεί είτε γίνει είτε δε γίνει κάποιο exception. Μάλιστα, ο κώδικας του `finally` θα εκτελεστεί ακόμα και εάν δε συλληφθεί κάποια εξαίρεση. Από τη στιγμή που ούτως ή άλλως ο κώδικας που υπάρχει μετά τα `catch` θα εκτελεστεί, η χρήση του `finally` επιβάλλεται κυρίως όταν το πρόγραμμα τερματίζεται νωρίτερα, π.χ. με κάποιο `return` ή μετά από κάποιο σφάλμα εξαίρεσης που δεν συλλαμβάνεται σε αντίστοιχο `catch`, όπως προαναφέρθηκε. Σε μια τέτοια περίπτωση, θα εκτελεστεί το `finally`-block και μετά θα τερματιστεί η εκτέλεση του προγράμματος.

- Εκτός από τις εξαιρέσεις που προκύπτουν αυτόματα όταν κάτι "πάει στραβά" σε κάποιο πρόγραμμα, ο προγραμματιστής έχει τη δυνατότητα να προκαλέσει ο ίδιος κάποια εξαίρεση. Ο τρόπος είναι ο εξής:

```
throw new ExceptionType("κείμενο περιγραφής της εξαίρ.");
```

Συνήθως μεταβιβάζουμε (προκαλούμε) κάποια εξαίρεση εάν συμβεί κάτι συγκεκριμένο, *δηλαδή συνήθως το `throw` βρίσκεται μέσα σε κάποια συνθήκη (π.χ., `if`)*. Στην ουσία, με το "`throw new`" δημιουργείται ένα αντικείμενο της κλάσης `ExceptionType` (του παραπάνω παραδείγματος) το οποίο παίρνει όνομα στο αντίστοιχο `catch`-block.

Άρα, γενικά, το ολοκληρωμένο block χειρισμού είναι: *`try-throw-catch`*:

```
try
{
   // δηλώσεις
   throw new ExceptionTypeX("δικό μας κείμενο");
   // δηλώσεις
}
catch(ExceptionType1 e1)
{
   // κάνε αυτό αν συμβεί η εξαίρεση ExceptionType1
}
.
.
.
catch(ExceptionTypeX eX)
{
   // κάνε αυτό αν συμβεί η εξαίρεση ExceptionTypeX
}
.
.
.
[finally
{
      // κάνε αυτό είτε συμβεί είτε δε συμβεί κάποια
      // από τις παραπάνω εξαιρέσεις
}]
```

- Τα `ExceptionTypeX` είναι συγκεκριμένοι τύποι εξαιρέσεων της Java, π.χ., `ArrayIndexOutOfBoundsException`, `NullPointerException`, `IOException`, `ClassNotFoundException`, `FileNotFoundException`, κτλ.

- Υπάρχουν μέθοδοι που επιστρέφουν περιγραφή της κάθε εξαίρεσης, π.χ.:

```
try
{
    ...
}
catch(ArrayIndexOutOfBoundsException e)
{
   System.out.println(e.toString()); // σύντομη περιγραφή εξαίρεσης
   System.out.println(e.getMessage()); // αναλυτική περιγρ. εξαίρεσης
}
```

Προσοχή! Αν μια εξαίρεση δε συμβαίνει μέσα σε `try {...}` ή αν δε συλλαμβάνεται στη συνέχεια σε κάποιο `catch(...){ }`, είναι σαν να μην υπάρχει, από άποψη χειρισμού (handling).

Παράδειγμα "try-throw-catch"

```
import java.util.*;

public class ExceptionsExample1
{
   public static void main(String [] args)
   {
      int donuts, milkGlasses;
      double donutsPerGlass;
      Scanner input = new Scanner(System.in);
      try
      {
         System.out.println("Πόσα donuts?");
         donuts = input.nextInt();
         System.out.println("Πόσα ποτήρια γάλα?");
         milkGlasses = input.nextInt();
         if (milkGlasses < 1)
            throw new Exception("Exception: No milk!");
         donutsPerGlass = donuts / (double)milkGlasses;
         System.out.println("Donuts per glass = " + donutsPerGlass);
      }
      catch (Exception e)
      {
         System.out.println(e.getMessage());
      }
      System.out.println("End of program.");
   }
}
```

Με inputs 3 και 2:

→ 3
→ 2
Donuts per glass: 1.5
End of program.

Με inputs 3 και 0:

→ 3
→ 0
Exception: No milk!
End of program.

Στη δεύτερη εκτέλεση του προγράμματος (με δεδομένα 3 και 0), δεν εκτελείται η εντολή `System.out.println("Donuts per glass = " + donutsPerGlass);` αφού στην ακριβώς προηγούμενη εντολή προκύπτει εξαίρεση, οπότε σταματάει η κανονική ροή του κώδικα και εκτελείται η εντολή του `catch`-block.

Δημιουργία κλάσης εξαίρεσης

Οι εξαιρέσεις είναι υποκλάσεις της κλάσης `Exception`. Άρα, όταν θέλουμε να δημιουργήσουμε μία δική μας κλάση εξαίρεσης, θα πρέπει να κληρονομούμε την κλάση `Exception`. Π.χ.:

```
public class DivideByZeroException extends Exception
{
   public DivideByZeroException()
   {
      super("Dividing by zero!");
   }
```

```
   public DivideByZeroException(String message)
   {
      super(message);
   }
}
```

και μία κλάση που χρησιμοποιεί αυτή την κλάση εξαίρεσης:

```
import java.util.*;
public class DivByZeroExample
{
   private int num, den;
   private double fraction;

   public void divide()
   {
      try
      {
         System.out.println("Αριθμητής?");
         Scanner input = new Scanner(System.in);
         num = input.nextInt();
         System.out.println("Παρανομαστής?");
         den = input.nextInt();
         if (den == 0)
             throw new DivideByZeroException();
         fraction = num / (double)den;
         System.out.println("Κλάσμα = " + fraction);
      }
      catch(DivideByZeroException e)
      {
         System.out.println(e.getMessage());
         secondChance();
      // υποθέτουμε ότι υπάρχει και μία μέθοδος secondChance
      // που ξαναζητάει την πιο προσεχτική είσοδο δεδομένων...
      }
   } // end of method devide

   public static void main(String [ ] args)
   {
      DivByZeroExample obj = new DivByZeroExample();
      obj.divide();
   }
}
```

[Στο συγκεκριμένο παράδειγμα, η μέθοδος `main` βρίσκεται στην ίδια την κλάση που βρίσκεται και η `divide`. Επειδή η `main` είναι `static`, δεν έχει άμεση πρόσβαση στη μέθοδο `divide` που δεν είναι `static`, οπότε για να την καλέσει πρέπει πρώτα να δημιουργήσει αντικείμενο (`obj`) της ίδιας της κλάσης στην οποία βρίσκεται. Αυτή η δημιουργία, όπως έχει αναφερθεί και σε προηγούμενο κεφάλαιο, είναι η μόνη λύση για την κλήση μη-`static` μεθόδων από `static` μεθόδους της ίδιας κλάσης. Εναλλακτικά, θα μπορούσε προφανώς να δηλωθεί ως `static` η μέθοδος `divide`.]

Δήλωση εξαιρέσεων σε μέθοδο:

Υπάρχουν περιπτώσεις που κάποια εξαίρεση μπορεί να θέλουμε να την κάνουμε `catch` μόνο εάν συντρέχουν κάποιοι λόγοι, που σημαίνει ότι μπορεί και να μην θέλουμε να την κάνουμε `catch` (δηλαδή να μη θέλουμε να τη διαχειριστούμε και να αφήσουμε το πρόγραμμα να τερματιστεί). Σε αυτές τις περιπτώσεις, ο κώδικας που μπορεί να μεταβιβάσει εξαίρεση μπαίνει *σε μια μέθοδο* (η

οποία στην ουσία *μεταβιβάζει (μεταθέτει) την ευθύνη του χειρισμού της εξαίρεσης* στη μέθοδο που την καλεί και) *η οποία δεν περιέχει* `try-catch` *block*, αλλά:

- *δηλώνει* στον ορισμό της (προειδοποιεί) ότι πιθανόν θα προκαλέσει κάποια εξαίρεση → αυτό γίνεται με την *πρόταση* `throws`
- *καλείται* μέσα σε ένα `try`-block (σε μια άλλη μέθοδο προφανώς) το οποίο φυσικά έχει και το αντίστοιχο `catch` (εάν τελικά θέλουμε να διαχειριστούμε την εξαίρεση – διαφορετικά η μέθοδος καλείται χωρίς `try-catch`)

Παράδειγμα:

```
import java.util.*;
public class ExceptionsExample2
{
   Scanner input = new Scanner(System.in);
   public int getInt() throws Exception
   {
      System.out.println("Πληκτρολόγησε έναν ακέραιο:");
      int n = input.nextInt();
      return n;
   }

   public void divide()
   {
      int n1=0, n2=1, n3=0;
      try
      {
         n1 = getInt();
         n2 = getInt();
         n3 = n1/n2;
      }
      catch(Exception e)
      {
         System.out.println(e.toString());
      }
      System.out.println(n1 + "/" + n2 + "=" + n3);
   }
}
```

Η μέθοδος `divide()` καλείται από τη `main` μέθοδο, στην κλάση εφαρμογής. Στη `main` λοιπόν:

```
ExceptionsExample2 ex2 = new ExceptionsExample2();
ex2.divide();
```

Αν τα inputs είναι π.χ. 22 και 3, τότε το πρόγραμμα θα τυπώσει:

```
22/3=7
```

Αν όμως τα inputs είναι π.χ. 22 και w, θα τυπώσει:

```
[InputMismatchException]
22/1=0
```

ενώ αν τα inputs είναι 22 και 0, τότε θα τυπώσει:

```
[ArithmeticException: / by zero]
22/0=0
```

Στη δεύτερη περίπτωση, ο κώδικας πετάει εξαίρεση γιατί αντί για ακέραιος δόθηκε χαρακτήρας (w) για το `n2`. Η εξαίρεση προκαλείται κατά την κλήση της μεθόδου `getInt()`, η οποία όντως προειδοποιούσε ότι μπορεί να συνέβαινε κάτι τέτοιο, και συλλαμβάνεται από το `catch`, το οποίο τυπώνει το σύντομο μήνυμα. Ο κώδικας συνεχίζει να εκτελείται, οπότε τυπώνονται οι τιμές των `n1`, `n2` και `n3`. Η τιμή του `n1` είναι αυτή που δόθηκε από το χρήστη, η τιμή του `n2` (1) είναι η αρχική του τιμή, αφού η εντολή απόδοσης τιμής στη μεταβλητή αυτή δεν εκτελέστηκε, επειδή συνέβη η εξαίρεση, και το ίδιο ισχύει και για το `n3` (όταν συμβαίνει κάποιο exception, η ροή του κώδικα διακόπτεται και πηγαίνει στο αντίστοιχο `catch`).

Αντίστοιχα, στην τρίτη περίπτωση, επιχειρείται να γίνει διαίρεση με το 0, οπότε ο κώδικας μεταβιβάζει την ανάλογη εξαίρεση (η εξαίρεση πλέον προκαλείται από εντολή της μεθόδου `divide()` και όχι από μεταβίβαση από τη μέθοδο `getInt()`), η οποία συλλαμβάνεται από το `catch`, το οποίο τυπώνει το σύντομο μήνυμα. Ο κώδικας συνεχίζει να εκτελείται, οπότε τυπώνονται οι τιμές που δόθηκαν για τα `n1` και `n2`, ενώ η τιμή 0 του `n3` είναι η αρχική του τιμή, αφού στην ουσία η διαίρεση δεν έγινε ποτέ.

Στο συγκεκριμένο παράδειγμα, το `catch` είναι πολύ γενικό για να συλλάβει και τις δύο διαφορετικές εξαιρέσεις που συμβαίνουν κατά την εκτέλεση.

Γενικός κανόνας: Αν μία μέθοδος μεταβιβάζει κάποια εξαίρεση (με την εντολή `throw`), τότε είτε πρέπει να περιέχει κάποιο `catch` για να τη συλλάβει, είτε πρέπει να δηλώνει στον ορισμό της ότι μεταβιβάζει κάποια εξαίρεση (με το `throws`).

Παράδειγμα:

```
public class CatchExample
{
   public static void main(String [] args)
   {
      CatchExample obj = new CatchExample();

      try
      {
         System.out.println("Trying");
         obj.myMethod();
         System.out.println("Trying after call.");
      }
      catch(Exception e)
      {
         System.out.println("Catching");
         System.out.println(e.getMessage());
      }
   }

   public void myMethod() throws Exception
   {
      System.out.println("Starting myMethod.");
      throw new Exception("From myMethod.");
   }
}
```

Το πρόγραμμα αυτό θα εκτυπώσει:

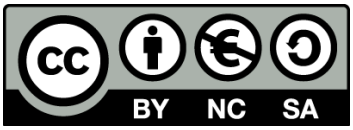
```
Trying
Starting myMethod.
Catching
From myMethod.
```

Σημειώματα

Σημείωμα Αναφοράς

Μιχάλης Δρακόπουλος. «Πληροφορική ΙΙ. Ενότητα 10: Exceptions handling (Χειρισμός εξαιρέσεων)». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: http://opencourses.uoa.gr/courses/MATH106/.

Σημείωμα Αδειοδότησης



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

- Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.