Αντικειμενοστραφής προγραμματισμός

Ενότητα 9

Χειρισμός σφαλμάτων

Γ' εξάμηνο - Τμ. Πληροφορικής - Ιόνιο Πανεπιστήμιο

Ρίγγας Δημήτρης
Μηχανικός Η/Υ & Πληροφορικής, MSc, PhD

riggas@ionio.gr

Χειρισμός σφαλμάτων(exception handling)

- Αντιμετώπιση εξαιρετικών σφαλμάτων που προκύπτουν κατά την εκτέλεση εντός προγράμματος
  - Π.χ.
    - Αναζήτηση εκτός ορίων πίνακα
    - Χειρισμός δεδομένων διαφορετικού τύπου
    - Απόπειρα κλήσης μεθόδου σε αναφορά null
    - ...
- Η αντιμετώπιση τέτοιων runtime errors, τα οποία δεν είναι εύκολο να ανιχνευθούν κατά τη μεταγλώττιση επιτρέπει τη δημιουργία προγραμμάτων
  - Ανθεκτικών σε σφάλματα
  - Που τερματίζονται με κομψό τρόπο

Παράδειγμα

- Ακέραια διαίρεση
  - Υπολογισμός πηλίκου
- *Τι μπορεί να πάει λάθος;*

```
import java.util.Scanner;

public class IntDivisor {

                                        // διαιρετέος,      διαιρέτης
   public static int quotient(int numerator, int denominator ) {
      return numerator/denominator;
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Δώσε διαιρετεό: ");
      int n = sc.nextInt();

      System.out.print("Δώσε διαιρέτη: ");
      int d = sc.nextInt();

      System.out.println("Πηλίκο: "+ quotient(n, d));

   }
}
```

Τι μπορεί να πάει λάθος σε ένα απλό πρόγραμμα;

Διαίρεση με μηδέν →

Να μη δοθεί ακέραιος →

```
import java.util.Scanner;

public class IntDivisor {

                                                // διαιρετέος,      διαιρέτης
   public static int quotient(int numerator, int denominator ) {
      return numerator/denominator;
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Δώσε διαιρετέό: ");
      int n = sc.nextInt();

      System.out.print("Δώσε διαιρέτη: ");
      int d = sc.nextInt();

      System.out.println("Πηλίκο: "+ quotient(n, d));

   }
}
```

Ποιο το αποτέλεσμα;

- Διαίρεση με μηδέν
  java IntDivisor
  Δώσε διαιρετέό: 3
  Δώσε διαιρέτη: 0
  Exception in thread "main" java.lang.ArithmeticException: / by zero
  at IntDivisor.quotient(IntDivisor.java:6)
  at IntDivisor.main(IntDivisor.java:18)

Exception

StackTrace

- Είσοδος διαφορετικού τύπου
  java IntDivisor
  Δώσε διαιρετέό: 5
  Δώσε διαιρέτη: hello
  Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:909)
  at java.util.Scanner.next(Scanner.java:1530)
  at java.util.Scanner.nextInt(Scanner.java:2160)
  at java.util.Scanner.nextInt(Scanner.java:2119)
  at IntDivisor.main(IntDivisor.java:16)

Δημιουργία Exception

- Οταν συμβαίνει ένα σφάλμα σε κάποια μέθοδο, η μέθοδος διακόπτει την κανονική της εκτέλεση, δημιουργεί ένα αντικείμενο τύπου Exception (συνηθέστερα υποκλάσης της) και το «πετάει» (throws).
- Το Exception αντικείμενο περιλαμβάνει πληροφορίες για το σφάλμα, όπως ο τύπος του και η κατάσταση του προγράμματος όταν συνέβη.
- Η Java περιμένει να βρει κώδικα ο οποίος θα πιάσει το Exception και είτε θα το διαχειριστεί είτε θα το «πετάξει» εκ νέου
  - Αναζητά στην τρέχουσα μέθοδο και στη συνέχεια σε όλες τις μεθόδους στη στοίβα εκτέλεσης μέχρι και τη main
  - Αν κανείς δεν πιάσει και διαχειριστεί ένα Exception τότε η JVM τυπώνει το μήνυμα σφάλματος που περιλαμβάνει:
    - Τον τύπο του Exception
    - Πιθανά μηνύματα
    - Που συνέβη το σφάλμα

Πώς προγραμματίζουμε για την αντιμετώπιση exceptions;

- *(Γιατί να το κάνουμε;)*
  - *Το πρόγραμμά μας μπαίνει σε μια αβέβαια κατάσταση αν δεν διαχειριστούμε σφάλματα που συμβαίνουν*

- Εντολές try … catch
- try block
  - Κώδικας που παρακολουθείται για εμφάνιση σφαλμάτων
- catch block
  - Χειριστής εξαίρεσης

Πώς χρησιμοποιούνται

```
...
try {
   // κώδικας που παρακολουθείται
   // πιθανό να προκύπτουν σφάλματα
   // εκτέλεσης τα οποία δημιουργούν
   // κάποια Exceptions
}
catch (ExceptionTypeA ex) {
   // χειρισμός ExceptionTypeA
   // ή υπο-κλάσεών της
}
catch (ExceptionTypeB ex) {
   // χειρισμός ExceptionTypeB
   // ή υπο-κλάσεών της
}
```

```
try {
        System.out.print("Δώσε διαιρετέ: ");
        int n = sc.nextInt();

        System.out.print("Δώσε διαιρέτη: ");
        int d = sc.nextInt();

        System.out.println("Πηλίκο: "+ quotient(n, d));
        exceptionFlag = false;
}
catch (ArithmeticException aex) {
        System.out.println("\nΣυνέβη ένα
                        ArithmeticException: "+ aex);
        System.out.println("Δοκιμάστε πάλι.
            Θυμηθείτε! Δεν διαιρούμε με το μηδέν.");
}
catch (java.util.InputMismatchException imex) {
        System.out.println("\nΣυνέβη ένα
            InputMismatchException: "+ imex);
        System.out.println("Δοκιμάστε πάλι... και
            αυτή τη φορά δώστε δύο ΑΚΕΡΑΙΟΥΣ!\n");
        sc.nextLine(); //Παραβλέπουμε τη
                        //λάθος είσοδο.
}
```

Τι περιέχει κάθε μπλοκ κώδικα

- try block
  - Κώδικας ο οποίος ίσως οδηγήσει σε σφάλμα
  - Κώδικας ο οποίος δεν πρέπει να εκτελεστεί εάν νωρίτερα υπάρξει σφάλμα
    - Ο κώδικας από το σημείο του σφάλματος έως το catch αγνοείται όταν συμβαίνει κάποιο σφάλμα
    - Μετά το χειρισμό του Exception από κάποιο catch block η εκτέλεση προγράμματος δεν επιστρέφει στο try block αλλά συνεχίζει μετά το τελευταίο catch block
      - Αν υπήρχαν local μεταβλητές εντός του try block, αυτές είναι πλέον εκτός εμβέλειας
- catch block
  - Κάθε catch block (μπορεί να υπάρχουν πολλά) πιάνει ένα τύπο Exceptions (και τις υπο-κλάσης αυτού) και πραγματοποιεί κατάλληλους χειρισμούς
  - Μετά από κάθε try block πρέπει να ακολουθεί ένα τουλάχιστον catch block (ή ένα finally block που θα δούμε πιο κάτω)

Περίπτωση finally

- Σε ορισμένες περιπτώσεις κάποιος κώδικας πρέπει να εκτελεστεί είτε η εκτέλεση του try block ήταν ομαλή ή ακόμη και αν προέκυψε κάποιο Exception.
  - Π.χ. κλείσιμο κάποιον πόρων συστήματος
- Για να αποφύγουμε επανάληψη τέτοιου κώδικα εντός του try block και όλων των catch blocks χρησιμοποιούμε το finally block

Παράδειγμα

```
try {
            System.out.print("Δώσε διαιρετέο: ");
            int n = sc.nextInt();

            System.out.print("Δώσε διαιρέτη: ");
            int d = sc.nextInt();

            System.out.println("Πηλίκο: "+ quotient(n, d));
            exceptionFlag = false;
}
catch (ArithmeticException aex) {
            System.out.println("\nΣυνέβη ένα ArithmeticException: "+
                                                aex);
            System.out.println("Δοκιμάστε πάλι. Θυμηθείτε! Δεν
                                                διαιρούμε με το μηδέν.\n");
        }
catch (java.util.InputMismatchException imex) {
            System.out.println("\nΣυνέβη ένα InputMismatchException: "+
                                                imex);
            System.out.println("Δοκιμάστε πάλι... Και αυτή τη φορά
                                                δώστε δύο ΑΚΕΡΑΙΟΥΣ!\n");
            sc.nextLine();
}
finally {
            System.out.println("Μη ξεχνάτε ότι το σύνολο των ακεραίων
δεν είναι κλειστό ως προς τη διαίρεση, μπορεί να υπάρχει και
υπόλοιπο!");
}
```

Προειδοποίηση για πιθανά Exceptions

- Ορθή προγραμματιστική πρακτική:
  - Οι μέθοδοι να πιάνουν τα Exceptions που μπορεί να συμβούν εντός τους και να τα διαχειρίζονται
  - ‘Η να προειδοποιούν για Exceptions τα οποία μπορεί να συμβούν

- Χρήση λέξης κλειδί throws
  - Δηλώνει μια λίστα από Exceptions που μια μέθοδος πιθανόν να πετάξει

Παράδειγμα

- Το throws δηλώνεται μετά την *υπογραφή* της μεθόδου και πριν τα άγκιστρα του block κώδικά της

```
public static int quotient(int numerator,
                    int denominator )
                    throws ArithmeticException
  { // διαιρετέος, διαιρέτης
     return numerator/denominator;
  }
```

Σκόπιμο πέταγμα Exception

- Δεν καλούμαστε απλά πιάσουμε Exceptions
  - Μπορούμε και να πατάμε Exceptions όποτε κρίνουμε ότι κάτι μη διαχειρίσιμο πρόκειται να συμβεί
- Χρήση εντολής throw
- Το ίδιο μπορεί να γίνει και εντός ενός catch block το οποίο δεν μπορεί πλήρως να διαχειριστεί ένα σφάλμα (ξανα-πετάμε το ίδιο Exception)
  - Το finally block εκτελείται!
- Οχι όμως εντός finally block

```
public static int quotient(int numerator
        , int denominator )
        throws ArithmeticException
{
    if (  denominator == 0 )
        throw new ArithmeticException
                ("Ο διαιρέτης δεν
                μπορεί να είναι μηδέν
                (/0)!");
    // διαιρετέος, διαιρέτης
    return numerator/denominator;
}
```

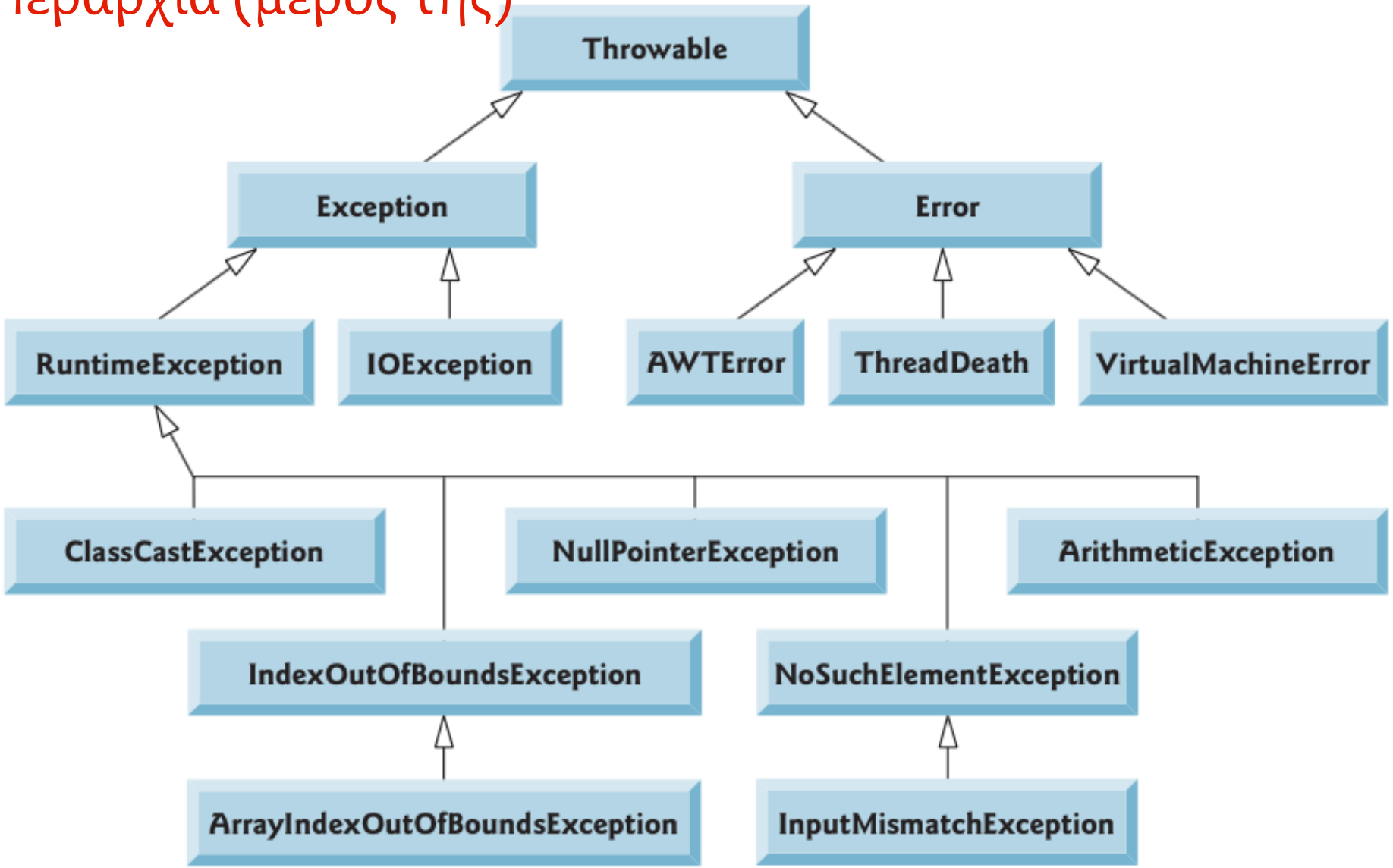

Κλάσεις
Throwable, Exception & Error

Throwable

- Η κλάση Trhowable είναι η υπερ-κλάση όλων των errors και exceptions στη Java
  - Μόνο αντικείμενά της ή υπο-κλάσεών της μπορούν να πεταχτούν από μια εντολή throw και μπορούν να είναι παράμετροι σε μια catch

- Εχει δυο απογόνους
  - Error
    - Εξαιρετικά σπάνια σφάλματα εντός της Virtual Machine
    - Τα προγράμματα δεν μπορούν συνήθως (και δεν πρέπει) να προσπαθούν να τα «πιάσουν», πχ OutOfMemoryError
  - Exception
    - Εξαιρετικές καταστάσεις εντός του προγράμματος
    - Τα προγράμματα μπορούν και οφείλουν να διαχειρίζονται

Ιεραρχία (μέρος της)



Ελεγχόμενα & μη-ελεγχόμενα (check & unchecked) Exceptions

- Η Java διακρίνει μεταξύ των Exceptions σε ελεγχόμενα και μη-ελεγχόμενα
- Για τα ελεγχόμενα ο compiler απαιτεί η μέθοδος
  - είτε να τα «πιάνει» και να τα διαχειρίζεται
  - ή να τα δηλώνει (throws)

- Μη-ελεγχόμενα
  - Κλάση και υπο-κλάσεις της java.lang.RuntimeException
  - Κλάση και υπο-κλάσεις της java.lang.Error
- Ελεγχόμενα
  - Υπο-κλάσης της Exception που δεν είναι υπο-κλάσεις της RuntimeException

Δημιουργία νέου τύπου Exception

- Αξιοποιώντας την κληρονομικότητα μπορούμε να δημιουργήσουμε νέες κλάσεις που επεκτείνουν την Exception ή κάποια υπο-κλάση της

- Ενας νέος τύπος MyCustomException συνήθως παρέχει τέσσερις constructors:

  public MyCustomException ()

  - Δημιουργεί ένα νέο MyCustomException χωρίς κάποιο μήνυμα λάθους

  public MyCustomException (String message)

  - Δημιουργεί ένα νέο MyCustomException με κάποιο μήνυμα λάθους

  MyCustomException (String message, Throwable cause)

  - Δημιουργεί ένα νέο MyCustomException χωρίς κάποιο μήνυμα λάθους και αιτία κάποιο προηγούμενο Throwable αντικείμενο

  MyCustomException (Throwable cause)

  - Δημιουργεί ένα νέο MyCustomException με αιτία κάποιο προηγούμενο Throwable αντικείμενο

Ενα Exception ως πηγή πληροφοριών

- Μέσω των μεθόδων:

  public String getMessage()
  - Returns the detail message string of this throwable.

  public Throwable getCause()
  - Returns the cause of this throwable or null if the cause is nonexistent or unknown. (The cause is the throwable that caused this throwable to get thrown.)

  public void printStackTrace()
  - Prints this throwable and its backtrace to the standard error stream.


Πλεονεκτήματα
χειρισμού με Exceptions

Πλεονεκτήματα χειρισμού σφαλμάτων με Exceptions

```
errorCodeType readFile {
   initialize errorCode = 0;

   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
      if (theFileDidntClose && errorCode == 0) {
         errorCode = -4;
      } else {
         errorCode = errorCode and -4;
      }
   } else {
      errorCode = -5;
   }
   return errorCode;
}
```

Διαχωρισμός λειτουργικού κώδικα από κώδικα χειρισμού σφαλμάτων

```
readFile {
   try {
      open the file;
      determine its size;
      allocate that much memory;
      read the file into memory;
      close the file;
   } catch (fileOpenFailed) {
      doSomething;
   } catch (sizeDeterminationFailed) {
      doSomething;
   } catch (memoryAllocationFailed) {
      doSomething;
   } catch (readFailed) {
      doSomething;
   } catch (fileCloseFailed) {
      doSomething;
   }
}
```

Πλεονεκτήματα χειρισμού σφαλμάτων με Exceptions

- Τα σφάλματα προχωρούν σε προηγούμενες μεθόδους στη στοίβα εκτέλεσης μέχρι κάποιος να ενδιαφερθεί να τα «πιάσει»
  - Ο χειρισμός σφαλμάτων γίεται εκεί που έχει ενδιαφέρον και με το τρόπο που εκεί κρίνεται ορθός

- Μπορεί να γίνει ομαδοποίηση του χειρισμού σφαλμάτων
  - Πχ τα MalformedURLException και FileNotFoundException μπορούν από κοινού να «πιαστούν» από ένα catch IOException (κοινή υπερ-κλάση) το οποίο διαχειρίζεται όλες τις περιπτώσεις που ένα αναμενόμενο αρχείο εισόδου δεν είναι διαθέσιμο

  - Προσοχή όμως στην υπερβολική γενίκευση, πχ catch Exception

Εργαστήριο #8

Τι μάθαμε;

Εργαστήριο #8– Ανακεφαλαίωση

- Διεπαφές
  - Υλοποίηση java.lang.Comparable interface
    - Αυτόματοποίηση διαδικασίας ταξινόμησης λίστας αντικειμένων

Εργαστήριο #9

Τι θα μάθουμε;

Πηγές

- Paul Dietel & Harvey Dietel, Java How to Program, 10/e
- Γρηγόρης Τσουμάκας, Αντικειμενοστρεφής Προγραμματισμός – Ενότητα 11 – Χειρισμός σφαλμάτων
- https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html