ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Χειρισμός Εξαιρέσεων
Exceptions Handling

Αθ. Ανδρούτσος

Λάθη (Errors) κατά την εκτέλεση

- Έχουμε συνηθίσει να γράφουμε κώδικα σε ιδανικά περιβάλλοντα όπου τυχόν αρχεία είναι στη σωστή θέση, το δίκτυο μας δεν διακόπτεται, το JVM μας έχει αρκετή μνήμη, κλπ.
- Δεν είναι πάντα όμως τα πράγματα τόσο ιδανικά. Σε πολλές περιπτώσεις δημιουργούνται μη ομαλές καταστάσεις κατά τη διάρκεια εκτέλεσης ενός προγράμματος, με συνέπεια τη **δημιουργία λαθών (errors)** από τα οποία θα θέλαμε να κάνουμε **recover**

Εξαιρέσεις (1)

- Τα σφάλματα που προκύπτουν κατά τη διάρκεια εκτέλεσης ενός προγράμματος στην Java ονομάζονται **εξαιρέσεις (exceptions)**
- ***Κάθε εξαίρεση σχετίζεται με μία κλάση*** που έχει οριστεί στην Java που μπορεί να παρέχει πληροφορία σχετικά με το συγκεκριμένο σφάλμα
- Οι κλάσεις που αντιπροσωπεύουν εξαιρέσεις είναι υποκλάσεις της κλάσης **Exception**

Κατηγορίες εξαιρέσεων

- Οι εξαιρέσεις στην Java ανήκουν σε δύο βασικές κατηγορίες:
  - **Checked**
    - Άμεσοι απόγονοι της κλάσης **Exception**. Αναπαριστούν exceptions **που μπορεί να προβλεφθεί ότι μπορεί να συμβούν**. Επομένως, **ελέγχονται κατά τη διάρκεια της μεταγλώττισης και** είναι υποχρεωτικό να τις χειριστούμε**.** Για παράδειγμα: *FileNotFoundException*, *IOException*, *SQLException*.
  - **Unchecked**
    - Άμεσοι απόγονοι της κλάσης **RunTimeException** που είναι υποκλάση της Exception. Αναπαριστούν exceptions **που αντιστοιχούν σε προγραμματιστικά λάθη και επομένως** δεν μπορούν να προβλεφθούν. **Δεν ελέγχονται κατά τη διάρκεια της μεταγλώττισης** αν γίνονται handle και επομένως δεν είναι υποχρεωτικό να τις χειριστούμε (handle). Για παράδειγμα: ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException.

Error class

- Εκτός από την κλάση Exception η Java παρέχει και την κλάση **Error**. Και η Exception και η Error class είναι υποκλάσεις της κλάσης **Throwable** που είναι η υπερκλάση όλων των exceptions και errors στην Java
- Η κλάση Error και οι υποκλάσεις της αντιστοιχούν σε λάθη που προκαλούνται από προβλήματα στα **system resources**, όπως **VMError** με υποκλάσεις **OutOfMemoryError** (που συνήθως προκαλείται από το heap Memory και μεγάλες σύνθετες δομές δεδομένων ή την αδυναμία το Garbage Collector να κάνει reclaim τη μνήμη στο Heap), **StackOverflowError** (που προκαλείται όταν το Stack memory έχει γίνει exhaust συνήθως από αέναες αναδρομικές κλήσεις, αναδρομικών μεθόδων)

Checked Exceptions (1)

- Τα Checked Exceptions είναι άμεσοι απόγονοι της κλάσης Exception. **Ελέγχονται κατά τη διάρκεια της μεταγλώττισης αν έχουν γίνει handle και επομένως είναι υποχρεωτικό να τις χειριστούμε (handle).** Για παράδειγμα: *IOException, FileNotFoundException, ParseException*
- **IOException.** Μία μέθοδος δημιουργεί (throws) IOException ή κάποια υποκλάση της IOException όταν συμβεί κάποιο λάθος κατά το **Input/Output**. Τέτοιες εξαιρέσεις περιλαμβάνουν εφαρμογές με αρχεία και γενικά data streams (java.io package) ή διαδικτυακές εφαρμογές (java.net package). Για παράδειγμα αν ένας Scanner συνδεθεί με αρχείο που δεν υπάρχει τότε δημιουργείται **FileNotFoundException** που είναι υποκλάση της IOException

Checked Exceptions (2)

- **ParseException**. Όταν διαβάζουμε ένα String για να δημιουργήσουμε ένα object, όπως Date object με συγκεκριμένη μορφή, π.χ. dd/mm/yyyy αλλά διαβάζουμε ένα string με διαφορετική μορφή

Checked Exceptions (3)

- Όταν στο σώμα μιας μεθόδου δημιουργείται ένα checked exception θα πρέπει:
  - είτε η μέθοδος στην επικεφαλίδα της να κάνει **throws το Exception,** ή
  - Είτε να το χειρίζεται (handle) το Exception με μία δομή **try/catch** που θα δούμε στα επόμενα
  - Ή και τα δύο παραπάνω, δηλαδή η μέθοδος χειρίζεται το exception με try/catch και το κάνει και **rethrow**, οπότε κάνει και throws στην επικεφαλίδα της

Unchecked Exceptions (1)

- Άμεσοι απόγονοι της κλάσης **RunTimeException** που είναι υποκλάση της Exception. Αναπαριστούν exceptions **που αντιστοιχούν σε προγραμματιστικά λάθη**. Δεν ελέγχονται κατά τη διάρκεια της μεταγλώττισης αν γίνονται handle και επομένως δεν είναι υποχρεωτικό να τις χειριστούμε. Για παράδειγμα: ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException
- **NullPointerException**. Δημιουργείται όταν καλούμε μεθόδους αντικειμένων και τα αντικείμενα δεν έχουν γίνει new, οπότε είναι null και επομένως δεν μπορούμε να κάνουμε invoke μεθόδους null αντικειμένων και δημιουργείται NullPointerException. Για παράδειγμα αν έχουμε **String s = null**; Και μετά if (**s.equals("AUEB")**) { }

Unchecked Exceptions (2)

- **ArrayIndexOutOfBoundsException**. Δημιουργείται όταν αναφερθούμε σε θέση ενός πίνακα που δεν υπάρχει. Όταν για παράδειγμα σε μία for αναφερθούμε σε **arr[i], όπου το i είναι ίσο με arr.length**, συνήθως επειδή σε μία for έχουμε i <= arr.length ενώ θα έπρεπε να πάει μέχρι τη θέση i < arr.length
- **NumberFormatException**. Δημιουργείται κατά τη **μετατροπή ενός string σε αριθμό**, όπως από τις wrapper κλάσεις Interger.parseInt(str), Double.parseDouble(str), κλπ. **Όταν το string δεν είναι σε κατάλληλη μορφή δηλαδή ακέραιος αριθμός**, ή δεκαδικός αντίστοιχα (δεν έχει δηλαδή αριθμητική αξία, αλλά άλλους χαρακτήρες), δημιουργείται NumberFormatException

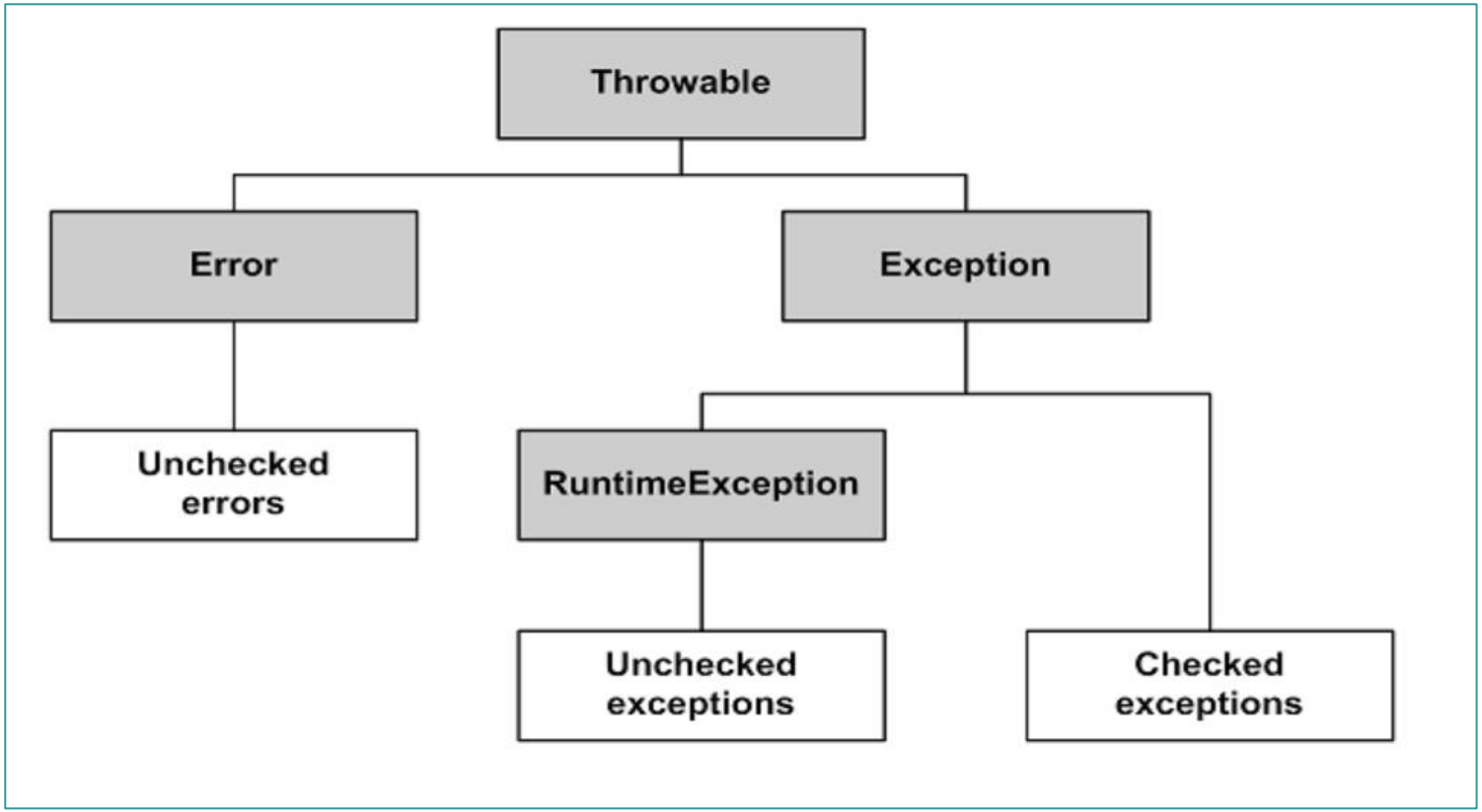
Unchecked Exceptions (3)

- **InputMismatchException**. Όταν προσπαθούμε να διαβάσουμε με **Scanner.nextInt()** ή **Scanner.nextDouble()** κλπ. αριθμητικά δεδομένα που δεν είναι όμως σε κατάλληλη μορφή, π.χ. **αντί για ακέραιος ή δεκαδικός αριθμός δοθεί στον Scanner κάποιο string** με χαρακτήρες μη αριθμητικούς, τότε δημιουργείται InputMismatchException
- **ArithmeticException**. Δημιουργείται όταν προσπαθήσει ένα πρόγραμμα να κάνει για παράδειγμα μία διαίρεση με το μηδέν μεταξύ int ή long

Unchecked Exception (4)

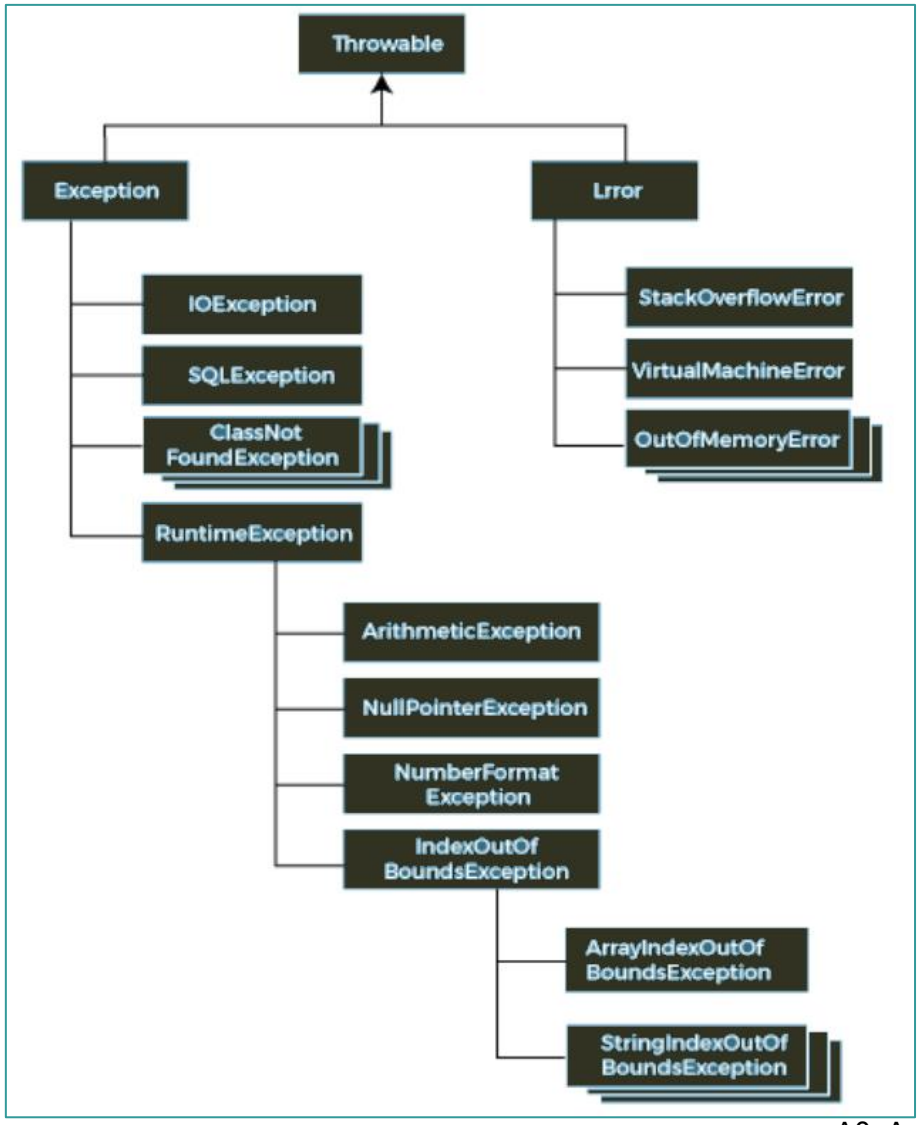
- **IllegalArgumentException**. Δημιουργείται όταν σε μία μέθοδο περνάμε ως πραγματική/ες παράμετρο/ους τιμές μη λογικά επιτρεπτές. Π.χ. η Thread.sleep(-12) δημιουργεί IllegalArgumentException γιατί δεν μπορεί να δεχτεί αρνητικές τιμές ως milliseconds
- **IllegalStateException**. Δημιουργείται όταν η τιμή μία μεταβλητής ενός αντικειμένου δεν είναι σε συνεπές state (τιμή) και κατά την κλάση μία μεθόδου αυτού του αντικειμένου δημιουργείται IllegalStateException. Π.χ. Όταν σε ένα iterator καλέσουμε την remove() χωρίς να έχουμε κάνει την 1η φορά .next()

Ιεραρχία εξαιρέσεων (1)



- Η Java παρέχει τριών τύπων throwables: checked exceptions, runtime exceptions και errors

Ιεραρχία εξαιρέσεων (2)



- Γενικά τα checked exceptions χρησιμοποιούνται σε συνθήκες που λογικά αναμένουμε ότι μπορούμε να κάνουμε recover
- Τα runtime exceptions είναι γενικά non-recoverable μιας και στη μεγάλη πλειοψηφία τους οφείλονται σε προγραμματιστικά λάθη
- Θα δούμε στα επόμενα πως μπορούμε να χειριζόμαστε εξαιρέσεις

Χειρισμός εξαιρέσεων (1)

- Δεν θα θέλαμε και δεν πρέπει ένα πρόγραμμα να σταματάει την εκτέλεσή του απότομα
- Η απότομη και ανεξέλεγκτη διακοπή ενός προγράμματος είναι σημαντικό πρόβλημα
- Ιδανικά θα θέλαμε να ***αντιμετωπίζουμε*** τις εξαιρέσεις, ώστε να μη διακόπτονται τα προγράμματα δηλαδή να κάνουμε **recover και να δίνουμε ένα μήνυμα στον χρήστη**

Χειρισμός εξαιρέσεων (2)

- Recoverable exceptions είναι τα checked exceptions
- Τα unchecked exceptions (runtime exceptions), δηλαδή exceptions που αφορούν προγραμματιστικά λάθη στον κώδικά μας δεν είναι recoverable. Απλά θα πρέπει να ανευρίσκονται και να διορθώνονται τα προγραμματιστικά λάθη
- Υπάρχουν ωστόσο περιπτώσεις που τα unchecked exceptions μπορούν να γίνουν recover, όπως το NumberFormatException ή InputMismatchException, αλλά καλύτερα σε αυτές τις περιπτώσεις να χρησιμοποιούμε, **όχι exceptions αλλά state testing**

Εξαιρέσεις και Χειρισμός (1)

- Καταρχάς θα πρέπει να απαντήσουμε στο ερώτημα, γιατί χρειαζόμαστε τον μηχανισμό των εξαιρέσεων (των checked exceptions)
- Ας υποθέσουμε ότι μία μέθοδος (π.χ. η main) καλεί μία άλλη μέθοδο, και σε αυτή την 'άλλη' μέθοδο δημιουργείται ένα checked exception
- Τότε **θα πρέπει η caller μέθοδος να ενημερωθεί για το Exception**, ώστε και αυτή από την πλευρά της να το χειριστεί κατάλληλα

Εξαιρέσεις και Χειρισμός (2)

- Ο χειρισμός ενός checked exception πρέπει να γίνει τόσο στην ίδια την μέθοδο (π.χ. rollback ή logging) όσο και στην caller μέθοδο
- Επίσης ο χειρισμός ενός Exception περιλαμβάνει τελικά και την εμφάνιση ενός κατάλληλου μηνύματος προς τον χρήστη

Εξαιρέσεις και Χειρισμός (2)

- Ο χειρισμός ενός exception σε μία μέθοδο περιλαμβάνει:
  - την **τεχνική αντιμετώπιση του Exception,** δηλαδή να **μη διακοπεί το πρόγραμμα** αλλά και να επαναφέρουμε σε μία πρότερη κατάσταση, π.χ. σε ένα unit of work (transaction) να κάνουμε rollback αν κάτι δεν πάει καλά,
  - Την **καταγραφή του exception** με Loggers, ώστε να υπάρχει ιχνηλασιμότητα
  - Την επαναμετάδοση του Exception προς τον caller (**rethrow**), ώστε και ο caller να ενημερωθεί για το failure και να το αντιμετωπίσει και αυτός από την πλευρά του
  - Καθώς επίσης και την **πληροφορία που θα δώσουμε στον χρήστη** αν η μέθοδος αφορά επικοινωνία με χρήστη

Εξαιρέσεις και Χειρισμός (3)

- Η επαναμετάδοση ενός checked exception προς τον caller μπορεί να γίνει με throws στην επικεφαλίδα της μεθόδου ή με try/catch και rethrow
- Επίσης θα πρέπει όταν δημιουργείται ένα exception να αποδεσμεύουμε τυχόν πόρους (resources, όπως π.χ. αρχεία) που έχουμε δεσμεύσει, όπως θα τα αποδεσμεύαμε αν δεν είχε συμβεί το exception

Εξαιρέσεις και Χειρισμός (4)

- Στο επίπεδο των μεθόδων που επιστρέφουν τιμές **μπορούμε να αποφεύγουμε το μηχανισμό των exceptions** για **κοινές καταστάσεις λάθους**, **επιστρέφοντας μη έγκυρες τιμές**, όπως -1 κατά την αναζήτηση της θέσης ενός στοιχείου που δεν υπάρχει σε πίνακα ή null όταν επιστρέφουμε σύνθετους τύπους
- **Δεν είναι τόσο καλό να επιστρέφουμε nulls** γιατί μπορεί να δημιουργήσουν NullPointerException στον caller (αν για παράδειγμα τον γράφει αρχάριος προγραμματιστής!), οπότε **θα μπορούσε σε αυτές τις περιπτώσεις να επιστρέφουμε μία wrapper κλάση των nulls που είναι η κλάση Optional**, που μπορεί να περιέχει είτε την έγκυρη τιμή ή null

Unchecked exceptions και χειρισμός (1)

- Τα unchecked exceptions δεν πρέπει να τα χειριζόμαστε (αν τα χειριζόμασταν θα ήταν σαν να αποκρύπτουμε (mask) ένα προγραμματιστικό λάθος) αλλά απλώς να διορθώνουμε τα προγραμματιστικά λάθη που τα προκαλούν
- Για παράδειγμα **να ελέγχουμε με if (state-testing) όταν είναι να διαιρέσουμε με μηδέν ή για nulls, να προσέχουμε τις for** σε πίνακες να μην πάνε έξω από τα όρια των πινάκων

Exceptions και Debugging (Αποσφαλμάτωση)

- Στις περιπτώσεις που έχουμε exceptions θα πρέπει να γίνεται και μία διαδικασία αποσφαλμάτωσης (debugging) δηλαδή μία διαδικασία εύρεσης και διόρθωσης unchecked exceptions
- Σε αυτή την περίπτωση θα θέλαμε πληροφορίες όπως ποια μέθοδος κάλεσε την μέθοδο, η οποία μπορεί να κάλεσε με τη σειρά της άλλη μέθοδο, κλπ. μέχρι ποια μέθοδος δημιούργησε το exception, το λεγόμενο **stack trace,** δηλαδή όλη τη στοίβα κλήσεων στο stack ξεκινώντας από το top του stack μέχρι τη μέθοδο που ξεκίνησε το stack των κλήσεων

Λογικά λάθη

- Τα λογικά λάθη, δηλαδή τα λάθη όπου επιστρέφονται λάθος αποτελέσματα, δεν πιάνονται όλα τα λάθη με exceptions, ούτε διακόπτεται το πρόγραμμα
- Γι' αυτό παράλληλα με την συγγραφή του κώδικα θα πρέπει να γράφονται και σενάρια ελέγχου (test cases) τα οποία θα πρέπει να ελέγχονται και όταν υπάρχουν λάθη να αποσφαλματώνονται (debug)
- Τα σενάρια ελέγχου μπορεί να είναι manual ή καλύτερα automated με κώδικα
- Όταν ανευρίσκονται λάθη θα πρέπει να αποσφαλματώνονται εισάγοντας breakpoints στον κώδικα ή στις μεθόδους που δημιουργούν το λάθος

Χειρισμός εξαιρέσεων - Δομή try/catch

- Για να χειριστούμε (handle) μία εξαίρεση βάζουμε τον κώδικα που μπορεί να προκαλέσει εξαίρεση μέσα στο ***try*** και τις εντολές αντιμετώπισης της εξαίρεσης μέσα στο ***catch.*** Το ***finally*** εκτελείται πάντα**.** Στο **finally** εισάγουμε εντολές αποδέσμευσης πόρων

- ```
  try
  {
      κώδικας που μπορεί να προκαλέσει εξαίρεση
  } catch (ExceptionType e)
  {
      εντολές χειρισμού της εξαίρεσης
  } finally {
      εντολές αποδέσμευσης (close) resources

  }
  ```
  ```

Αποσφαλμάτωση

- Τα προγραμματιστικά λάθη (unchecked exceptions) δεν χρειάζεται (***και δεν πρέπει***) να τα κάνουμε try/catch
- Η Java χωρίς try/catch διακόπτει το πρόγραμμα και εμφανίζει το exception stack trace
- Επομένως κατά τη διάρκεια των ελέγχων μπορούμε να εντοπίζουμε και να διορθώνουμε τα λάθη αυτά

Unchecked Exceptions

Θα δούμε στη συνέχεια παραδείγματα **unchecked exceptions** και πως κάνουμε debugging με τον debugger του IntelliJ

Arithmetic Exception (1)


```java
package gr.aueb.cf.testbed.ch8;

public class ArithmeticExceptionApp {

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 0;
        int result = 0;

        result = num1 / num2;
        System.out.println(result);
    }
}
```

```
Run:  gr.aueb.cf.testbed.ch8.ArithmeticExceptionApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Program Files\J
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at gr.aueb.cf.testbed.ch8.ArithmeticExceptionApp.main(ArithmeticExceptionApp.java:10)

Process finished with exit code 1
```

- Το πρόγραμμα ορίζει δύο ακεραίους, αριθμητή και παρονομαστή με 0, οπότε αποτυγχάνει η διαίρεση.

Arithmetic Exception (2)


```java
package testbed.ch8;

import java.util.Scanner;

/**
 * java Arithmetic Exception
 */
public class ArithmeticExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int numerator = 0;
        int denominator = 0;
        int result = 0;

        System.out.println("Please insert a numerator and a denominator");
        numerator = in.nextInt();
        denominator = in.nextInt();

        result = numerator / denominator;

        System.out.printf("%d / %d = %d", numerator, denominator, result);
    }
}
```

- Το πρόγραμμα λαμβάνει δύο ακεραίους, αριθμητή και παρονομαστή από τον χρήστη και στη συνέχεια πραγματοποιεί διαίρεση

Arithmetic Exception (2)

- Δίνοντας ο χρήστης ως παρονομαστή το 0 γίνεται προσπάθεια διαίρεσης ακεραίων με το 0 στη γραμμή 20 του προγράμματος
- Το πρόγραμμα διακόπτει την εκτέλεσή του απότομα καθώς δημιουργήθηκε ένα unchecked exception, το ArithmeticException

Debugging (1)

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help LengthIsEmptyApp.java - ArithmeticExceptionApp.java

CodingFactory › src › testbed › ch8 › ArithmeticExceptionApp

ArithmeticExceptionApp

AdditionsApp.java | BinaryToDecimalApp.java | BitwiseOperatorsApp.java | CombinationsFourApp.java | Array15App.java | ArithmeticExceptionApp.java

```
1  package testbed.ch8;
2
3  import java.util.Scanner;
4
5  /**
6   * java Arithmetic Exception
7   */
8  public class ArithmeticExceptionApp {
9
10     public static void main(String[] args) {
11         Scanner in = new Scanner(System.in);
12         int numerator = 0;
13         int denominator = 0;
14         int result = 0;
15
16         System.out.println("Please insert a numerator and a denominator");
17         numerator = in.nextInt();
18         denominator = in.nextInt();
19
20         result = numerator / denominator;
21
22         System.out.printf("%d / %d = %d", numerator, denominator, result);
23     }
24 }
```

- Εισάγουμε ένα breakpoint (με κλικ λίγο δεξιά από τον αριθμό 20, στη γραμμή 20) και κλικ στο κουμπί debug (πάνω δεξιά)

Debugging (2)


```
10  public static void main(String[] args) {   args: []
11      Scanner in = new Scanner(System.in);   in: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=4][match valid=true][need input=fa
12      int numerator = 0;   numerator: 10
13      int denominator = 0;   denominator: 0
14      int result = 0;   result: 0
15
16      System.out.println("Please insert a numerator and a denominator");
17      numerator = in.nextInt();
18      denominator = in.nextInt();   in: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=4][match valid=true][need input=false][sou
19
20      result = numerator / denominator;   denominator: 0    result: 0    numerator: 10
21
22      System.out.printf("%d / %d = %d", numerator, denominator, result);
23  }
```

```
Debug:  ArithmeticExceptionApp
Debugger  Console
"main"@1 in...in": RUNNING
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
main:20, ArithmeticExceptionApp (testbe
p args = {String[0]@963} []
in = {Scanner@964} "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=4][match valid=true][need input=false][source closed=false][skipped=false][group separ ... View
01 numerator = 10
01 denominator = 0
01 result = 0
Switch frames from anywhere in the IDE wit...
```

- Στο debugger pane δίνονται οι τιμές των μεταβλητών στο breakpoint, ενώ στο παράθυρο του κώδικα δίνονται οι τιμές των μεταβλητών από την αρχή του κώδικα μέχρι το breakpoint. Παρατηρούμε ότι στη γραμμή του breakpoint η τιμή του denominator είναι 0, οπότε και διαπιστώσαμε την πηγή του λάθους.

Handling Runtime Exceptions

- Μερικές φορές δεν είναι σαφές αν ένα runtime exception είναι recoverable ή όχι
- Εν προκειμένω το ArithemeticException προήλθε από την αλληλεπίδραση με τον χρήστη και δυνητικά θα μπορούσε να γίνει recover, δεν είναι προγραμματιστικό λάθος
- Θα μπορούσαμε να το χειριστούμε με try-catch **ή καλύτερα με state-testing (if)**

Try/catch


```java
package testbed.ch8;

import java.util.Scanner;

/**
 * java Arithmetic Exception with try/catch
 */
public class ArithmeticException2 {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int numerator = 0;
        int denominator = 0;
        int result = 0;

        while (true) {
            try {
                System.out.println("Please insert a numerator (0 for quit) and a denominator");
                numerator = in.nextInt();
                if (numerator == 0) {
                    break;
                }
                denominator = in.nextInt();
                result = numerator / denominator;
                System.out.printf("%d / %d = %d\n", numerator, denominator, result);
            } catch (ArithmeticException e) {
                System.out.println("Error. Denominator should not ne zero");
            }
        }
        System.out.println("Thanks for using the app");
    }
}
```

- Μπορούμε να χρησιμοποιήσουμε try/catch εισάγοντας μέσα στο try τον κώδικα που δημιουργεί το ArithmenticException και στη συνέχεια το κάνουμε catch
- Στο catch χειριζόμαστε το exception. Εδώ απλά δίνουμε μήνυμα. **Καλύτερα θα ήταν να έχουμε ελέγξει ήδη την τιμή του παρονομαστή πριν φτάσουμε στο exception**

State-dependent μέθοδοι

- Όταν έχουμε state-dependent μεθόδους (π.χ. in.nextInt() ή Integer.parseInt()) ή state-dependent expressions όπως διαίρεση int / int όπου ο παρονομαστής μπορεί να είναι 0, και μπορεί να κληθούν κάτω από μη-προβλέψιμες συνθήκες, καλύτερα θα ήταν να παρείχαμε **μία state-testing μέθοδο (true/false), που να μας υποδεικνύει πότε είναι κατάλληλο να καλούμε τη μέθοδο**

State-testing


```java
30      /**
31       * State-testing method for zero
32       *
33       * @param num    the number to check for
34       * @return       true, if the num is zero, false otherwise
35       */
36      public static boolean isZero(int num) {
37          return num == 0;
38      }
39  }
```

- Ελέγχει ένα αριθμό αν είναι μηδέν

Έλεγχος με state-testing


```java
package testbed.ch8;

import java.util.Scanner;

public class ArithmeticExceptionWithStateTest {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int numerator = 0;
        int denominator = 0;
        int result = 0;

        while (true) {
            System.out.println("Please insert a numerator and a denominator");
            numerator = in.nextInt();
            denominator = in.nextInt();

            if (isZero(denominator)) {      // if (denominator == 0)
                System.out.println("Please insert a valid denominator");
                continue;
            }
            result = numerator / denominator;
            System.out.printf("%d / %d = %d\n", numerator, denominator, result);
            if (result == 1) break; // a random condition for exit
        }

        System.out.println("Thanks for using the app");
    }
```

- Μπορούμε να ελέγξουμε με τη χρήση της state-testing μεθόδου που λειτουργεί με μία δομή
- Πιο απλά ακόμα θα ήταν αν απλά κάναμε:
- if (dominator == 0)
- Αυτή είναι η ουσία του state-testing, να ελέγξουμε από πριν αν η τιμή που μπορεί να δημιουργήσει πρόβλημα είναι έγκυρη

ArrayIndexOutOfBounds

- Στη συνέχεια θα ακολουθήσει ένα παράδειγμα της εξαίρεσης ***ArrayIndexOutOfBoundsException*** που συμβαίνει όταν προσπαθούμε να έχουμε πρόσβαση σε στοιχεία ενός πίνακα αλλά το index που δίνουμε είναι έξω από τα όρια του πίνακα
- Για παράδειγμα, συμβαίνει όταν έχουμε ένα πίνακα 3 θέσεων από 0 έως 2 και προσπαθούμε να προσπελάσουμε το στοιχείο με index 3, το οποίο όμως δεν υπάρχει
- Την εξαίρεση αυτή που είναι unchecked δεν πρέπει να την κάνουμε try/catch γιατί δεν είναι σωστό να συνεχιστεί το πρόγραμμα 'ομαλά'. Αντίθετα ***θα πρέπει να διορθώσουμε τον κώδικα***

ArrayIndexOutOfBoundsException


```
1  package testbed.ch8;
2
3  /**
4   * Throws ArrayIndexOutOfBoundsException
5   */
6  public class ArrayIndexOutOfBoundsExceptionApp {
7
8      public static void main(String[] args) {
9          int[] arr = {1, 2, 3};
10
11         for (int i = 0; i <= arr.length; i++) {
12             System.out.print(arr[i] + " ");
13         }
14     }
15 }
```

- Υπάρχει αναφορά μέσα στην print() στο στοιχείο app[3] (όταν το i γίνει arr.length, δηλ. 3) που δεν υπάρχει μιας και ο πίνακας έχει 3 θέσεις από 0 – 2
- Το πρόβλημα είναι στη for και στο = της i <= arr.length
- Θα έπρεπε να είναι χωρίς =

```
Run: ArrayIndexOutOfBoundsExceptionApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ ID
1 2 3 Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
    at testbed.ch8.ArrayIndexOutOfBoundsExceptionApp.main(ArrayIndexOutOfBoundsExceptionApp.java:12)

Process finished with exit code 1
```

- Επειδή η for εκτελείται μέχρι <= arr.length, στη γραμμή 12 η θέση arr[i] δεν υπάρχει όταν το i γίνει ίσο με arr.length και δημιουργείται ArrayIndexOutOfBoundsException

Debugging


```
6  public class ArrayIndexOutOfBoundsExceptionApp {
7
8      public static void main(String[] args) {   args: []
9          int[] arr = {1, 2, 3};   arr: [1, 2, 3]
10
11         for (int i = 0; i <= arr.length; i++) {   i: 3
12             System.out.print(arr[i] [ArrayIndexOutOfBoundsException] + " ");   arr: [1, 2, 3]   i: 3
13         }
14     }
15 }
```

Debug: ArrayIndexOutOfBoundsExceptionApp

Debugger | Console

"main"@1 in...in": RUNNING

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

main:12, ArrayIndexOutOfBoundsExcept

Resume Program F9

args = {String[0]@702} []

arr = {int[3]@703} [1, 2, 3]

i = 3

arr.length = 3

arr[i] = java.lang.IndexOutOfBoundsException : Invalid array range: 3 to 3

- Εισάγοντας breakpoint στη γραμμή 12 και πατώντας debug και μετά διαδοχικά Resume μέχρι να αναπαράγουμε το λάθος, παρατηρούμε ότι για i = 3 το arr[i] δεν υπάρχει, οπότε συνάγουμε ότι θα πρέπει να διορθώσουμε το range της for

Ορθό πρόγραμμα

BinaryToDecimalApp.java | BitwiseOperatorsApp.java | CombinationsF

```
1  package testbed.ch8;
2
3  /**
4   * Throws ArrayIndexOutOfBoundsException
5   */
6  public class ArrayIndexOutOfBoundsExceptionApp {
7
8      public static void main(String[] args) {
9          int[] arr = {1, 2, 3};
10
11         for (int i = 0; i < arr.length; i++) {
12             System.out.print(arr[i] + " ");
13         }
14     }
15 }
```

- Διορθώνουμε τη συνθήκη τέλους της for σε < arr.length, οπότε το πρόγραμμα είναι ορθό

NullPointerException

- Επίσης, η εξαίρεση NullPointerException συμβαίνει όταν χρησιμοποιούμε ***μία σύνθετη δομή (array, String, object) χωρίς να την κάνουμε new (οπότε παραμένει null)*** και στη συνέχεια χρησιμοποιούμε μία ιδιότητα ή μέθοδο της δομής
- Και σε αυτή την περίπτωση θα πρέπει να βρούμε το λάθος κατά τον έλεγχο και να διορθώσουμε τον κώδικα
- Αν ωστόσο μία αναφορά είναι possibly null και δεν είναι λάθος του κώδικα θα πρέπει από πριν να έχουμε χρησιμοποιήσει state-testing για null

NullPointerException (1)


```java
package testbed.ch8;

/**
 * Null Pointer Exception.
 */
public class NullPointerExceptionApp {

    public static void main(String[] args) {

        String s = null;

        if (s.equals("Coding")) System.out.println("Coding");
    }
}
```

- Όταν ένα String instance είναι null τότε δεν μπορούμε να καλέσουμε μεθόδους όπως equals(), length(), charAt() κλπ. γιατί δημιουργείται εξαίρεση NullPointerException

NullPointerException (2)


```
1   package testbed.ch8;
2
3   /**
4    * Null Pointer Exception.
5    */
6   public class NullPointerExceptionApp {
7
8       public static void main(String[] args) {
9
10          String s = null;
11
12          if (s.equals("Coding")) System.out.println("Coding");
13      }
14  }
```

- Δημιουργείται NullPointerException στη γραμμή 12, όταν κάνουμε dereference το s (access τη θέση που δείχνει ο pointer s) μιας και δεν μπορούμε να κάνουμε dereference ένα null pointer

```
Run:  NullPointerExceptionApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Program
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at testbed.ch8.NullPointerExceptionApp.main(NullPointerExceptionApp.java:12)

Process finished with exit code 1
```

Debugging


```
3   /**
4    * Null Pointer Exception.
5    */
6   public class NullPointerExceptionApp {
7
8       public static void main(String[] args) {   args: []
9
10          String s = null;   s: null
11
12          if (s.equals("Coding")) System.out.println("Coding");   s: null
13      }
14  }
15
```

Debug: NullPointerExceptionApp

Debugger | Console

"main"@1 in...in": RUNNING

Evaluate expression (Enter) or add a watch (Ctrl+S

main:12, NullPointerExceptionApp (testb

args = {String[0]@701} []

s = null

- Κάνοντας debugging παρατηρούμε την τιμή null στο s

Διόρθωση λάθους (1)


```java
package testbed.ch8;

/**
 * Null Pointer Exception.
 */
public class NullPointerExceptionApp {

    public static void main(String[] args) {

        String s = "A String";

        if (s.equals("Coding")) System.out.println("Coding");
    }
}
```

- Δίνουμε στο String μία τιμή

Null state-testing


```java
package gr.aueb.cf.ch8;

public class NullPointerApp {

    public static void main(String[] args) {
        String s;
        int secret = 11;

        s = getBravoOrNull(secret);

        if (s == null) {
            System.out.println("Not Lucky!");
            return;
        }

        // If s was null we would get NullPointerException
        // on s.equals(), if we had not checked before.
        if (s.equals("Bravo")) {
            System.out.println("Very Lucky");
            return;
        }
        System.out.println("Indifference!");
    }

    public static String getBravoOrNull(int secret) {
        final int SECRET = 12;

        if (secret != SECRET) return null;
        return "Bravo";
    }
}
```

- Η μέθοδος **getBravoOrNull** μπορεί να επιστρέψει string με τιμή "Bravo" ή null
- Η main ελέγχει για null πριν χρησιμοποιήσει το string για να ελέγξει για "Bravo"
- Αν δεν είχε ελέγξει για null και το string ήταν null, τότε στο s.equals("Bravo") θα είχαμε NullPointerException

Scanner input

- Έχουμε δει ότι με τον ***Scanner*** μπορούμε να διαβάζουμε πρωταρχικούς τύπους με τις ***nextInt()***, ***nextDouble()***, κλπ., καθώς και Strings με την next()
- Αν όμως πάμε να διαβάσουμε π.χ. με την nextInt() και ***ο χρήστης δώσει αντί για ακέραιο ένα String*** τότε δημιουργείται ***InputMismatchException***
- Η InputMismacthException είναι unchecked exception

InputMismatchException (1)


```java
package testbed.ch8;

import java.util.Scanner;

/**
 * Input mismatch exception when user inserts
 * a non-numeric string instead of integer.
 */
public class InputMismatchExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;

        System.out.println("Please insert an int");
        inputNum = in.nextInt();

        System.out.println("Input number: " + inputNum);
    }
}
```

- Η Scanner.nextInt() περιμένει να διαβάσει ένα integer
- Οτιδήποτε άλλο θα δημιουργήσει InputMismatchException

InputMismatchException (2)


```
Run:  InputMismatchExceptionApp
Please insert an int
coding
Exception in thread "main" java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at testbed.ch8.InputMismatchExceptionApp.main(InputMismatchExceptionApp.java:16)

Process finished with exit code 1
```

- Δίνοντας ένα string η .nextInt() αποτυγχάνει και δημιουργείται InputMismatchException

Try/catch


```java
package testbed.ch8;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * Input mismatch exception when user inserts
 * a non-numeric string instead of integer.
 */
public class InputMismatchExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;

        try {
            System.out.println("Please insert an int");
            inputNum = in.nextInt();
            System.out.println("Input number: " + inputNum);
        } catch (InputMismatchException e) {
            e.printStackTrace();
        }
    }
}
```

- Θα μπορούσαμε να κάνουμε handle το Exception μέσα σε μια try/catch
- Στην try δίνουμε τον κώδικα που μπορεί να δημιουργήσει το exception
- Στην catch δίνουμε τον κώδικα που κάνει handle το exception
- Εδώ απλά κάνουμε log το stack trace. Η e.printStackTrace() αποτυπώνει όλο το stack trace δηλαδή την ακολουθία κλήσεων των μεθόδων στο Stack μέχρι που συνέβη το exception

e.printStackTrace() (1)

- Η προηγούμενη try/catch απλά κάνει log στο standard err όπως θα γινόταν και χωρίς try/catch

```
Run:  InputMismatchExceptionApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Program Fi
Please insert an int
n
java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at testbed.ch8.InputMismatchExceptionApp.main(InputMismatchExceptionApp.java:18)

Process finished with exit code 0
```

e.printStackTrace() (2)


```
Run:  InputMismatchExceptionApp
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Program Fi
Please insert an int
n
java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at testbed.ch8.InputMismatchExceptionApp.main(InputMismatchExceptionApp.java:18)

Process finished with exit code 0
```

- Παρατηρούμε ότι η printStackTrace δίνει πολλή πληροφορία και χρησιμοποιείται κατά τη φάση του development για λόγους debugging. Όπως ακριβώς δημιουργούνται τα stack frames κατά τη φάση του invocation μίας μεθόδου από μία άλλη, έτσι εδώ βλέπουμε ότι αρχικά στη γραμμή 18 της main έγινε ένα invocation που μετά από διαδοχικές κλήσεις σε άλλες μεθόδους της κλάσης Scanner δημιούργησε το Exception

Try/catch μέσα σε while (1)


```java
/**
 * Input mismatch exception when user inserts
 * a non-numeric string instead of integer.
 */
public class InputMismatchExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;

        while (true) {
            try {
                System.out.println("Please insert an int");
                inputNum = in.nextInt();
                if (inputNum == 12) break;
                System.out.println("Input number: " + inputNum);
            } catch (InputMismatchException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Try/catch μέσα σε while (2)


```
Run:  InputMismatchExceptionApp
java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at testbed.ch8.InputMismatchExceptionApp.main(InputMismatchExceptionApp.java:19)
java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
Please insert an int
Please insert an int
Please insert an int
Please insert an int
```

- Μέσα στη while η nextInt() δεν μπορεί να καταναλώσει τη λέξη than, αφού δεν είναι int. Το αποτέλεσμα είναι, λόγω της while, συνεχώς να εμφανίζεται το μήνυμα λάθους και η προτροπή να δοθεί αριθμός

Try/catch μέσα σε while (3)


```java
/**
 * Input mismatch exception when user inserts
 * a non-numeric string instead of integer.
 */
public class InputMismatchExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;

        while (true) {
            try {
                System.out.println("Please insert an int");
                inputNum = in.nextInt();
                if (inputNum == 12) break;
                System.out.println("Input number: " + inputNum);
            } catch (InputMismatchException e) {
                in.nextLine();
                e.printStackTrace();
            }
        }
    }
}
```

- Εκτός από logging, αντιμετωπίζουμε και το exception καταναλώνοντας το string input με in.nextLine() ώστε να κάνουμε recover από το exception

Try/catch και μήνυμα προς τον χρήστη (2)


```java
/**
 * Input mismatch exception when user inserts
 * a non-numeric string instead of integer.
 */
public class InputMismatchExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;

        while (true) {
            try {
                System.out.println("Please insert an int");
                inputNum = in.nextInt();
                if (inputNum == 12) break;
                System.out.println("Input number: " + inputNum);
            } catch (InputMismatchException e) {
                in.nextLine();
                //e.printStackTrace();
                System.out.println("Error in input, please insert an int");
            }
        }
    }
}
```

- Κατά τη φάση του development δίνουμε πολλή πληροφορία με την printStackTrace() ώστε να κάνουμε debugging
- Κατά τη φάση του Production θα θέλαμε μηνύματα φιλικά προς τον χρήστη και όχι αναλυτική πληροφορία για το Exception κάτι το οποίο θα συνιστούσε και πρόβλημα ασφάλειας κατά τη φάση της παραγωγής

Έλεγχος με hasNextInt()


```java
/**
 * InputMismatchException with state-testing.
 */
public class InputMismatchException2App {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int inputNum = 0;
        final int MAGIC = 12;

        while (true) {
            System.out.println("Please insert an int");

            if (in.hasNextInt()) {
                inputNum = in.nextInt();
            } else {
                System.out.println("Please insert a valid int");
                in.nextLine();
                continue;
            }
            if (inputNum == MAGIC) break;
            System.out.println("Input num = " + inputNum);
        }
    }
}
```

- Όπως αναφέραμε στις περιπτώσεις state-dependent μεθόδων όπως η .nextInt() θα θέλαμε state-testing μεθόδους, που αποτελούν εναλλακτικό τρόπο ελέγχου
- Και πράγματι η Java μας παρέχει την hasNextInt() που ελέγχει αν το επόμενο token είναι int

NumberFormatException (1)

- Όταν μετατρέπουμε Strings σε ακεραίους με την ***Integer.parseInt()***, αν αντί για αριθμητικούς χαρακτήρες δώσουμε μη-αριθμητικούς χαρακτήρες δημιουργείται σφάλμα ***NumberFormatException***
- Ας δούμε στις επόμενες διαφάνειες πως το αντιμετωπίζουμε

NumberFormatException (2)


```java
package testbed.ch8;

import java.util.Scanner;

/**
 * NumberFormatException is thrown when
 * we parse a non-numeric token with
 * wrapper classes, such as Integer.parseInt()
 */
public class NumberFormatExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        String inputStr = "";

        System.out.println("Please insert an int");
        inputStr = in.nextLine();
        num = Integer.parseInt(inputStr);
        System.out.println("Input number: " + num);
    }
}
```

- Δημιουργείται όταν πάμε να κάνουμε parse με Integer.parseInt() ένα αλφαριθμητικό που δεν γίνεται evaluate σε int

NumberFormatException (3)


```
NumberFormatExceptionApp
Please insert an int
w
Exception in thread "main" java.lang.NumberFormatException Create breakpoint : For input string: "w"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at testbed.ch8.NumberFormatExceptionApp.main(NumberFormatExceptionApp.java:19)

Process finished with exit code 1
```

- Δίνοντας ένα αλφαριθμητικό, π.χ. w, δημιουργείται το exception και εμφανίζεται το stack trace

NumberFormatException (4)


```java
package testbed.ch8;

import java.util.Scanner;

/**
 * NumberFormatException is thrown when we parse a non-numeric
 * token with wrapper classes, such as Integer.parseInt()
 */
public class NumberFormatExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        String inputStr = "";

        try {
            System.out.println("Please insert an int");
            inputStr = in.nextLine();
            num = Integer.parseInt(inputStr);
            System.out.println("Input number: " + num);
        } catch (NumberFormatException e) {
            e.printStackTrace();
        }
    }
}
```

- Με try/catch θα μπορούσαμε να χειριστούμε το exception

NumberFormatException (5)


```
26      /**
27       * Returns true if the source string
28       * evaluates to integer.
29       *
30       * @param s      the source string
31       * @return       true, if the string evaluates to int,
32       *               false otherwise
33       */
34      public static boolean isInt(String s) {
35          try {
36              Integer.parseInt(s);
37              return true;
38          } catch (NumberFormatException e) {
39              return false;
40          }
41      }
42  }
```

Ωστόσο επειδή η Integer.parseInt() είναι state-dependent μέθοδος θα θέλαμε κάποια state-testing μέθοδο πριν κάνουμε το parse

- Η μέθοδος ***isInt(String s)*** ελέγχει αν μπορεί να μετατραπεί ένα String σε ακέραιο. Αν μετατρέπεται και δεν δημιουργηθεί NumberFormatException, τότε επιστρέφει true, αλλιώς γίνεται catch το Exception και επιστρέφει false

NumberFormatException (6)


```java
package testbed.ch8;

import java.util.Scanner;

/**
 * NumberFormatException is thrown when we parse a non-numeric
 * token with wrapper classes, such as Integer.parseInt()
 */
public class NumberFormatExceptionApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;
        String inputStr = "";

        System.out.println("Please insert an int");
        inputStr = in.nextLine();
        if (isInt(inputStr)) {
            num = Integer.parseInt(inputStr);
            System.out.println("Num: " + num);
        } else {
            System.out.println("Error in parsing. Please insert an integer");
        }
    }
```

- Με τη χρήση της state-testing μεθόδου isInt(), πρώτα ελέγχουμε αν το string είναι int και μετά κάνουμε parse

Finally block

- Υπάρχουν περιπτώσεις που ο κώδικας **μετά την try/catch** δεν εκτελείται (δείτε την επόμενη διαφάνεια)
- Αν εμείς θέλουμε να εκτελέσουμε ένα κομμάτι κώδικα **ΟΠΩΣΔΗΠΟΤΕ** είτε εκτελεστεί μόνο η try ή η catch, τότε θα πρέπει να χρησιμοποιήσουμε την ***finally*** που αποτελεί -αν χρειάζεται- μέρος της try/catch
- Συνήθως κάτι τέτοιο χρειάζεται όταν έχουμε να κλείσουμε πόρους όπως ο Scanner

Finally block


```java
package testbed.ch8;

/**
 * try/catch/finally.
 */
public class FinallyApp {

    public static void main(String[] args) throws Exception {
        int num = 12;
        try {
            if (num == 12) throw new Exception();
        } catch (Exception e) {
            return;
        } finally {
            System.out.println("Coding Factory is always executed!");
        }

        System.out.println("This piece of code is not always executed");
    }
}
```

- Επειδή έχουμε return μέσα στην catch, αν συμβεί το exception, ο κώδικας μετά την try/catch δεν εκτελείται
- Αντίθετα, ο κώδικας μέσα στη finally εκτελείται πάντα

Finally και Resources (1)

- Το πιο συνηθισμένο use case για τη χρήση της finally είναι όταν θέλουμε να κάνουμε close τα resources που έχουμε δεσμεύσει (π.χ. αρχεία που έχουμε ανοίξει) ακόμα κι αν συμβεί εξαίρεση

Finally και Resources (2)

- Για παράδειγμα η εντολή Scanner in = new Scanner(System.in); Δεσμεύει ένα resource (το standard input)
- Η εντολή in.close() αποδεσμεύει (κλείνει) τον Scanner και παρότι δεν την έχουμε δει , εκτελείται έμμεσα κάθε φορά που τελειώνει η εκτέλεση ενός προγράμματος που χρησιμοποιεί Scanner

finally – πάντα εκτελείται


```java
package testbed.ch8;

import java.util.InputMismatchException;
import java.util.Scanner;

public class FinallyResourceApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;

        try {
            num = in.nextInt();
            System.out.println();
        } catch (InputMismatchException e) {
            e.printStackTrace();
        } finally {
            in.close();
        }
    }
}
```

- Στην finally εισάγουμε εντολές που θέλουμε να εκτελεστούν σε συνέχεια του προστατευμένου κώδικα
- Ο Scanner γίνεται close σε κάθε περίπτωση

Εμφωλιασμένη try/catch


```
6   /**
7    * try/catch/finally closing resources.
8    */
9   public class FinallyResourceApp {
10
11      public static void main(String[] args) {
12          Scanner in = new Scanner(System.in);
13          int num = 0;
14
15          try {
16              num = in.nextInt();
17              System.out.println();
18          } catch (InputMismatchException e) {
19              e.printStackTrace();
20          } finally {
21              try {
22                  in.close();
23              } catch (Exception e) {
24                  e.printStackTrace();
25              }
26
27          }
28      }
29  }
```

- Επειδή ***μπορεί να συμβεί Exception και μέσα στην finally*** εισάγουμε try/catch
- Αν για παράδειγμα ο Scanner είχε κλείσει από κάποια άλλη διεργασία η ***in.close()*** θα δημιουργούσε Exception, το οποίο κάνουμε catch

Ταυτόχρονη δημιουργία εξαιρέσεων

- Σε περιπτώσεις που δημιουργείται exception ***και στην try και στην finally***, προτεραιότητα δίνεται στην finally (στο πιο πρόσφατο exception)
- Αυτό δεν είναι σωστό γιατί σημαντικότερες είναι οι εξαιρέσεις στην try
- Για να αντιμετωπιστεί αυτό το πρόβλημα η Java παρέχει για την try/catch και τη μορφή try-with-resources

Try-with-resources

- Αυτόματη διαχείριση πόρων
  - Όταν στο πρόγραμμά μας χρησιμοποιούμε πόρους που πρέπει να κλείσουν όπως αρχεία ή αντικείμενα τύπου Scanner, μπορούμε να χρησιμοποιήσουμε ένα μηχανισμό αυτόματου κλεισίματος ώστε να μην χρειάζεται να γράφουμε την finally
  - Ο μεταγλωττιστής της Java αυτόματα παρέχει την finally με τα αντίστοιχα close, των resources που αναφέρονται στην επικεφαλίδα της try-with-resources

Αυτόματη διαχείριση πόρων


```java
package testbed.ch8;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * try/catch/finally closing resources.
 */
public class FinallyResourceApp {

    public static void main(String[] args) {
        // Scanner in = new Scanner(System.in);
        int num = 0;

        try (Scanner in = new Scanner(System.in)) {
            num = in.nextInt();
            System.out.println();
        } catch (InputMismatchException e) {
            e.printStackTrace();
        }
    }
}
```

- Ο Scanner ορίζεται στην επικεφαλίδα της try και στη συνέχεια κλείνει αυτόματα

>= JDK9


```java
package testbed.ch8;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * try/catch/finally closing resources.
 */
public class FinallyResourceApp {

    public static void main(String[] args) {
        int num = 0;
        Scanner in = new Scanner(System.in);

        try (in) {
            num = in.nextInt();
            System.out.println();
        } catch (InputMismatchException e) {
            e.printStackTrace();
        }
    }
}
```

- Σύμφωνα με το JDK Enhancement Proposal 213 (JEP 213) που είναι μέρος της Java 9 μπορούμε να δηλώνουμε έξω από την try/catch μεταβλητές που είναι final ή effectively final και να τις χρησιμοποιήσουμε μέσα στην try/catch
- Effectively final είναι οι μεταβλητές που δεν είναι final αλλά δεν αλλάζουν τιμή και δεν βρίσκονται στο αριστερό μέρος μίας εκχώρησης μέσα στην try ή στην catch

Try-with-resources

- Με την try-with-resources αν δημιουργηθούν ταυτόχρονες εξαιρέσεις τότε αντιμετωπίζεται η εξαίρεση στην try και όχι στην finally (που έχει δημιουργηθεί αυτόματα από τον μεταγλωττιστή)

User-defined Exceptions

- Εξαιρέσεις που ορίζονται από τον χρήστη
- Πρόκειται για εξαιρέσεις μπορεί ο προγραμματιστής να ορίσει και κληρονομούν από την κλάση Exception
- Θα δούμε αυτού του τύπου τις εξαιρέσεις στην ενότητα του Αντικειμενοστραφούς Προγραμματισμού

Εξαιρέσεις

- Μέσα σε μια try μπορούν να δημιουργηθούν διάφορες εξαιρέσεις
- Εξαιρέσεις δημιουργούνται με δύο τρόπους:
  - Αν δημιουργήσουμε εμείς μία εξαίρεση με throw new <Τύπος Εξαίρεσης>
  - Αν καλέσουμε μία μέθοδο που η ίδια δημιουργεί εξαίρεση

Μέθοδοι που δημιουργούν εξαιρέσεις (1)

- Για παράδειγμα η ***System.in.read()*** που είχαμε δει στο κεφάλαιο με το τύπο char, είχαμε δει ότι θα έπρεπε να γράψουμε το throws IOException στην επικεφαλίδα της main
- Ο λόγος είναι ότι αν πάμε στα Java docs της Oracle θα δούμε πως η read() throws IOException, που είναι checked exception και πρέπει **είτε να γίνει throws** από την επικεφαλίδα της μεθόδου **ή να γίνει try/catch**

Εξαιρέσεις – try/catch


```java
package testbed.ch8;

import java.io.IOException;

/**
 * Checked exceptions could be addressed
 * using try/catch
 */
public class ThrowsVsTryCatch {

    public static void main(String[] args) {
        try {
            char ch = (char) System.in.read();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- Γενικά υπάρχουν τρεις εναλλακτικοί τρόποι 'αντιμετώπισης' εξαιρέσεων
- **Ή κάνουμε try/catch** όπως στο παρόν παράδειγμα
- **Είτε κάνουμε throws**, χωρίς try/catch, όπως θα δούμε στην επόμενη διαφάνεια
- **Στις παραγωγικές εφαρμογές θα πρέπει να χρησιμοποιείται ένας συνδυασμός των δύο παραπάνω τρόπων**

Εξαιρέσεις - throws


```java
package testbed.ch8;

import java.io.IOException;

/**
 * Checked exceptions could be addressed
 * throwing the exception to the caller.
 */
public class ThrowsVsTryCatch {

    public static void main(String[] args) throws IOException {
        char ch = (char) System.in.read();
    }
}
```

- Η εξαίρεση που δημιουργείται από την System.in.read() γίνεται throws

Πολλαπλές Εξαιρέσεις (1)

- Στην ιεραρχία των εξαιρέσεων της Java, η πιο γενική εξαίρεση είναι η ***Exception*** που είναι η υπερκλάση όλων των τύπων εξαιρέσεων, ενώ πιο ειδικές είναι οι υποκλάσεις της Exception όπως η IOException και η FileNotFoundException που είναι υποκλάση της IOException
- Αν σε μια try θέλουμε να κάνουμε handle και τις γενικότερες και τις πιο ειδικές εξαιρέσεις τότε μπορούμε να κάνουμε πολλαπλά catch, ***από το ειδικότερο στο γενικότερο*** και όχι το αντίθετο
- Πάντα θα πρέπει δηλαδή να υπάρχει μία κατάταξη (order) των εξαιρέσεων από την ειδικότερη προς την γενικότερη υποκλάση

Πολλαπλές Εξαιρέσεις (2)


```
1  package testbed.ch8;
2
3  import java.io.File;
4  import java.io.FileNotFoundExceptio
5  import java.io.IOException;
6  import java.util.Scanner;
7
8  public class MultipleExceptionsApp
9
10     public static void main(String[
11         File file = new File ("C:\\
12
13         try (Scanner in = new Scann
14             char ch = (char) System
15             int num = in.nextInt();
16         } catch (Exception e) {
17             e.printStackTrace();
18         } catch (IOException e) {
19             e.printStackTrace();
20             System.err.println("Exc
21         } catch (FileNotFoundException e) {
22             e.printStackTrace();
23         }
24     }
25 }
```

Exception 'java.io.FileNotFoundException' has already been caught

Exception 'java.io.FileNotFoundException' has already been caught

Delete catch for 'java.io.FileNotFoundException' Alt+Shift+Enter

java.io

```
public class FileNotFoundException
extends java.io.IOException
```

Signals that an attempt to open the file denoted by a specified pa
This exception will be thrown by the FileInputStream, FileOu
RandomAccessFile constructors when a file with the specified
exist. It will also be thrown by these constructors if the file does e
reason is inaccessible, for example when an attempt is made to c
writing.

Since: 1.0

Author: unascribed

< 11 >

- Αν βάλουμε 1η στη σειρά των catch την Exception υπερκαλύπτει τις επόμενες δύο γιατί τις περιλαμβάνει
- Το ίδιο ισχύει και για την IOException που υπερκαλύπτει την FileNotFoundException
- Κάτι τέτοιο δημιουργεί compile error

Πολλαπλές Εξαιρέσεις (3)


```java
package testbed.ch8;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

public class MultipleExceptionsApp {

    public static void main(String[] args) {
        File file = new File ("C:\\tmp\\random-file.txt");

        try (Scanner in = new Scanner(file)) {
            char ch = (char) System.in.read();
            int num = in.nextInt();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Exception: " + e);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- Η κατάταξη πρέπει να είναι από το ειδικότερο προς το γενικότερο exception

Μέθοδοι και exceptions

- Όταν γράφουμε μία μέθοδο που αντιμετωπίζει ένα error condition έχουμε τρεις βασικές επιλογές:
  - Χρησιμοποιούμε το return value για να καταδείξουμε το success ή failure
  - Χρησιμοποιούμε μία 'return code' παράμετρο
  - Κάνουμε throw ένα exception

Return value (Success / Failure)


```java
/**
 * Returns the position in an array of a
 * num.
 *
 * @param arr   the input array.
 * @param num   the input num.
 * @return      the position in the array, if the num
 *              was found, -1 otherwise.
 */
public static int findNumPositionInArray(int[] arr, int num) {
    int positionToReturn = -1;

    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == num) {
            positionToReturn = i;
            break;
        }
    }
    return positionToReturn;
}
```

- Μπορούμε να χρησιμοποιούμε error codes ως returned values όταν η μέθοδος επιστρέφει τιμή και δεν είναι void και όταν η επιστρεφόμενη τιμή λάθους είναι έξω από τα έγκυρα όρια τιμών (distinguished value)
- Στο παράδειγμα, το -1 δεν είναι έγκυρη θέση πίνακα, και χρησιμοποιείται ως error code

Nulls / Optional


```
    * Returns a string that starts with a specific prefix.
    *
    * @param arr   the input array of strings.
    * @param str   the input prefix.
    * @return      the string in the array that matches the
    *              input prefix, null otherwise.
    */
   public static Optional<String> getStringStartsWith(String[] arr, String str) {
       if (arr == null || str == null) return Optional.empty();

       Optional<String> token = Optional.empty();

       for (String s : arr) {
           if (s.startsWith(str)) {
               token = Optional.of(str);
               break;
           }
       }
       return token;
   }
```

- Αν υπάρχει η πιθανότητα επιστροφής null καλύτερα να επιστρέφουμε Optional για να αποφύγουμε την πιθανότητα NullPointerException στον Caller

Διαφορετικά αν επιστρέφει null προϋποθέτει τον έλεγχο στην πλευρά του caller για null. Αν δεν γίνει έλεγχος στον caller, τότε μπορεί να συμβεί NullPointerException στον Caller

Optional (1)


```java
/**
 * Returns a copy of a given array of ints
 * Instead of null, it returns Optional.
 * Since the array is mutable we return not
 * just a reference but a copy of the array.
 *
 * @param arr       the source array
 * @return          an Optional, null or empty
 */
public static Optional<int[]> getCopy(int[] arr) {
    if (arr == null) return Optional.empty();
    return Optional.of(Arrays.copyOf(arr, arr.length));
}

/**
 * It echoes the given string back to the caller.
 * Since strings are immutable we can just get
 * the reference back.
 *
 * @param s     the source string
 * @return      the source string as Optional object
 */
public static Optional<String> getStringCopy(String s) {
    return Optional.ofNullable(s);
}
```

- Η **κλάση Optional** είναι wrapper κλάση που μπορεί να περιέχει έγκυρες τιμές ή και null
- Δημιουργήθηκε για να αποφύγουμε τα null pointer exceptions στον caller
- Με την **Optional.of()** κάνουμε typecast μια τιμή σε Optional
- Με **Optional.ofNullable()** η τιμή μπορεί να είναι και null

Optional (2)


```java
package testbed.ch8;

import java.util.Arrays;
import java.util.Optional;

public class MethodsErrorsOptionalsWithoutExceptions {

    public static void main(String[] args) {
        String s = "Athens";
        Optional<String> opt = getStringCopy(s);
        opt.ifPresent(System.out::println);
    }
```

- Η **ifPresent()** ελέγχει αν το optional δεν είναι null και εκτυπώνει με callback function. Στην Java, όπως θα δούμε, μπορούμε να έχουμε callback functions με το μηχανισμό των Lambda expressions ή όπως εδώ των method references (το **::** είναι τελεστής εμβέλειας)

Προβλήματα return values

- Τα return values όταν χρησιμοποιούνται ως error indicators έχουν δύο βασικούς περιορισμούς
  - Θα πρέπει ο caller να ελέγχει την επιστρεφόμενη τιμή, κάτι που μπορεί να μη γίνεται και να οδηγεί σε μη έλεγχο των λαθών
  - Πιθανά ο caller να μην θέλει να κάνει handle το error και να θέλει απλά να το κάνει propagate σε μία υψηλότερου επιπέδου μέθοδο (caller της αρχικής caller).

Exceptions

- Θα ήταν προτιμότερο λοιπόν να χρησιμοποιούμε το μηχανισμό των exceptions ώστε να μην είναι αναγκαίο ο caller να ελέγχει για μη έγκυρα return values και επίσης να μπορεί εύκολα να κάνει propagate το exception σε μία υψηλότερου επιπέδου μέθοδο για να χειριστεί το exception

Return code parameter

- Ο τρόπος να περνάνε τα error codes ως παράμετροι ή με παρόμοιο τρόπο μπορεί να χρησιμοποιηθεί σε Web based εφαρμογές όπου μπορούμε να επιστρέφουμε HTTP response status codes
- Διαφορετικά οδηγεί σε παρόμοια προβλήματα όπως και οι επιστρεφόμενες τιμές

Μέθοδοι και Exceptions

- Οι μέθοδοι που χειρίζονται error conditions θα πρέπει να κάνουν τα εξής στο catch (θα πρέπει να κάνουν try/catch/finally)
  - **Restore state (αν χρειάζεται)**. Κάνοντας κάποιες ενέργειες, π.χ. rollback αν πρόκειται για transactions
  - **Logging**. Γράφουν στο std error με e.printStackTrace() ή σε παραγωγικές εφαρμογές γράφουν σε logfiles
  - **Rethrow**, για να στείλουν το Exception εκεί από όπου έχουν κληθεί (caller)

Finally

- Όταν χρησιμοποιούνται πόροι που υλοποιούν το AutoClosable interface καλό θα ήταν να χρησιμοποιείται η **try with resources** (χωρίς finally)
- Ενώ όταν οι πόροι δεν είναι AutoClosable θα πρέπει να κλείνουν μέσα στη finally

Μέθοδοι / Exceptions (1)


```
18      public static char readChar() throws IOException {
19          try {
20              return (char) System.in.read();
21          } catch (IOException e) {
22              e.printStackTrace();        // Logging
23              throw e;                    // rethrow to caller
24          }
25      }
```

- Παρατηρούμε ότι η μέθοδος κάνει catch το exception και μετά το κάνει rethrow. Σε άλλες περιπτώσεις θα μπορούσε να γίνει restore το state
- Ο λόγος που κάνουμε try / catch και rethrow και δεν αφήνουμε το Exception να γίνει propagate **είναι γιατί θέλουμε να γράψουμε στα logs**

Μέθοδοι / Exceptions (2)


```java
package testbed.ch8;

import java.io.IOException;

public class MethodReadCharException {

    public static void main(String[] args) {
        char ch;

        try {
            ch = readChar();
            System.out.println(ch);
        } catch (IOException e) {
            System.out.println("char read error");  // user friendly message
        }
    }
```

- Η main που είναι ο caller της readChar κάνει και αυτή try/catch αφού η readChar() κάνει throws IOException
- Στο catch δίνει μήνυμα φιλικό προς τον χρήστη

Αλληλεπίδραση με χρήστη

- Στο στάδιο της παραγωγής ο χρήστης πρέπει να λαμβάνει φιλικά μηνύματα feedback όταν συμβαίνει ένα exception (όχι e.printStackTrace())
- Τα μηνύματα τα δίνει στον χρήστη όποια μέθοδος επικοινωνεί με τον χρήστη, στην περίπτωση του παραδείγματος η main
- Δεν είναι concern της readChar() να δίνει μηνύματα προς τον χρήστη

Μέθοδοι / Exceptions (1)


```
21  @   public static Scanner getFileDescriptor(String s) throws FileNotFoundException {
22          File fd = new File(s);
23
24          try {
25              return new Scanner(fd);
26          } catch (FileNotFoundException e) {
27              e.printStackTrace();
28              throw e;
29          }
30      }
```

- Παραπάνω η μέθοδος getFileDescriptor επιστρέφει ένα αντικείμενο τύπου Scanner. Μπορεί ωστόσο να δημιουργηθεί FileNotFoundException αν ο Scanner αντιστοιχηθεί σε αρχείο που δεν υπάρχει
- Οπότε, η μέθοδος κάνει try/catch και: 1) γράφει στο log για να καταγραφεί το exception και να υπάρχει ιχνηλάτηση, και 2) κάνει rethrow ώστε να ενημερωθεί ο caller

Main


```java
package gr.aueb.cf.ch8;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.InputMismatchException;
import java.util.Scanner;

public class TryWithResourcesApp {

    public static void main(String[] args) throws FileNotFoundException {
        int num = 0;
        File fd = new File("C:\\tmp\\hello.txt");

        try (Scanner in = new Scanner(fd)) {
            num = in.nextInt();
            System.out.println(num);
        } catch (InputMismatchException | FileNotFoundException e) {
            e.printStackTrace();
            throw e;
        }
    }
}
```

- Μπορούμε να έχουμε και λογικό Ή με | στην catch και να κάνουμε catch περισσότερα από ένα exceptions

State Testing


```java
package gr.aueb.cf.ch8;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class TryWithResourcesApp {

    public static void main(String[] args) throws FileNotFoundException {
        int num = 0;
        File fd = new File("C:\\tmp\\hello.txt");

        try (Scanner in = new Scanner(fd)) {
            if (in.hasNextInt()) {
                num = in.nextInt();
                System.out.println(num);
            } else {
                System.out.println("Error in input");
                in.nextLine();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            throw e;
        }
    }
```

- Επίσης, μπορούμε και με state-testing χωρίς να χρειάζεται να κάνουμε try / catch το Input Mismatch Exception

Μικρή Εργασία Κεφαλαίου

- Αναπτύξτε μία μέθοδο που εμφανίζει ένα μενού με επιλογές από 1 έως 4 (και 5 για Exit) καθώς και μία μέθοδο int getChoice() για να διαβάσετε το choice του χρήστη με Scanner
- Στη μέθοδο int getChoice() αν ο χρήστης δώσει μη-ακέραιο ελέγξτε κατάλληλα με state-testing (hasNextInt())
- Αναπτύξτε επίσης μία μέθοδο void printChoice(int choice) που εκτυπώνει την επιλογή. Αν η επιλογή δεν είναι μεταξύ 1-5 τότε δημιουργεί (throws) ένα IllegalArgumentException, το οποίο παρότι είναι RunTimeException, θα μπορούσαμε στη συγκεκριμένη περίπτωση να κάνουμε recover και να κάναμε try/catch στη main
- Διαμορφώστε τη main ώστε να κάνει catch το exception και να επικοινωνεί με τον χρήστη με κατάλληλο μήνυμα