Αντικειμενοστρεφής Προγραμματισμός
Διάλεξη – 9 : ΕΞΑΙΡΕΣΕΙΣ ΚΑΙ Ο ΧΕΙΡΙΣΜΟΣ ΤΟΥΣ
EXCEPTIONS & EXCEPTION HANDLING

Εξαιρέσεις (Exceptions)

- Γνωμικό της Μηχανικής Λογισμικού: το 80% της δουλειάς των μηχανικών κατά την υλοποίηση μιας εφαρμογής, καταλαμβάνει μόλις το 20% του χρόνου εκτέλεσης.
- Ο εντοπισμός και η αντιμετώπιση σφαλμάτων αποτελούν από τα πιο βασικά συστατικά μιας καλογραμμένης εφαρμογής.
- Η Java εφοδιάζει τους προγραμματιστές με έναν «κομψό» μεν, αποτελεσματικό δε μηχανισμό αντιμετώπισης σφαλμάτων, ο οποίος παράλληλα βοηθάει στην καλύτερη δόμηση και οργάνωση του κώδικα. Ο μηχανισμός αυτός είναι γνωστός ως μηχανισμός χειρισμού εξαιρέσεων (exception handling).

Τι είναι οι εξαιρέσεις?? (1/2)

- Με τον όρο ΕΞΑΙΡΕΣΗ αναφερόμαστε στη σπάνια αλλά πιθανή να συμβεί περίπτωση, όπου ο κώδικάς μας δε λειτουργεί κανονικά λόγω μιας απρόοπτης κατάστασης. Μία εξαίρεση δεν είναι ένα σφάλμα στον κώδικα αλλά μία ανεπιθύμητη κατάσταση που μπορεί να προκύψει από άλλον λόγο, όπως για παράδειγμα στα εξής σενάρια:
  - Κάποιο αρχείο που θέλουμε να ανοίξουμε δεν βρέθηκε
  - Κάποια βάση δεδομένων στην οποία θέλουμε να συνδεθούμε είναι offline
  - Μία δικτυακή σύνδεση απέτυχε
- Στα παραπάνω σενάρια, έχουμε ως δεδομένο πως ο κώδικας που εκτελεί τη συγκεκριμένη διεργασία είναι σωστός και η αποτυχία ολοκλήρωσής της οφείλεται σε κάποιον εξωγενή παράγοντα.

Τι είναι οι εξαιρέσεις?? (2/2)

- Παραδείγματα:
  - στην πρώτη περίπτωση ο χρήστης έγραψε λάθος το όνομα του αρχείου και γι αυτό δε βρέθηκε,
  - στη δεύτερη μπορεί ο administrator να έχει «ρίξει» τον server offline,
  - στην Τρίτη περίπτωση μπορεί κάποιο καλώδιο να είναι προβληματικό.
- Κάθε φορά που συμβαίνει ένα αντίστοιχο γεγονός κατά την εκτέλεση ενός προγράμματος Java, γίνεται thrown («πετάγεται») μία εξαίρεση.
- Αυτός είναι και ο λόγος που η συγκεκριμένη κατηγορία σφαλμάτων ονομάζονται εξαιρέσεις, μιας και πρόκειται για περιπτώσεις όπου η σωστή λειτουργία του κώδικα αποτελεί τον κανόνα, ενώ η περίπτωση σφάλματος αποτελεί την εξαίρεση.

Χειρισμός Εξαιρέσεων

- Για σενάρια σαν αυτά που προαναφέρθηκαν στα οποία προκύπτει εξαίρεση, το πρόγραμμα διακόπτεται συνήθως λόγω έλλειψης απαραίτητων δεδομένων.
- Ο μη κατάλληλος χειρισμός τέτοιου είδους σφαλμάτων έχει συνήθως ως αποτέλεσμα τον βίαιο και μη ελεγχόμενο τερματισμό του προγράμματος.
- Στη Java, ο χειρισμός εξαιρέσεων είναι ο μηχανισμός που μας επιτρέπει να χειριστούμε σφάλματα κατά την εκτέλεση του προγράμματός μας τα οποία συνήθως οφείλονται σε εξωγενείς παράγοντες.

Βασικά χαρακτηριστικά χειρισμού εξαιρέσεων

- Ένα από τα βασικά χαρακτηριστικά του μηχανισμού χειρισμού εξαιρέσεων είναι πως δίνει στον προγραμματιστή τη δυνατότητα να εντοπίσει την ακριβή αιτία που προκάλεσε το πρόβλημα, καθώς και το σημείο του κώδικα όπου προκλήθηκε.
- Εξίσου σημαντικό είναι όμως το γεγονός πως αποφεύγεται η βίαιη διακοπή της εφαρμογής.
- Ο πιο συνηθισμένος χειρισμός είναι να προβληθεί κάποιο μήνυμα στον χρήστη και το πρόγραμμα να τερματίσει ελεγχόμενα και με ομαλό τρόπο.
- Έτσι, με τη βοήθεια του μηχανισμού χειρισμού εξαιρέσεων οι προγραμματιστές είναι σε θέση να γράφουν ανθεκτικές (fault tolerant) σε σφάλματα εφαρμογές

Μηχανισμός χειρισμού εξαιρέσεων (1/4)

- Η βασική αρχή του είναι η μεταφορά της ροής σε κάποιον κατάλληλο χειριστή, στην περίπτωση που προκύψει εξαίρεση. Αυτό γίνεται με τη βοήθεια ενός συνδυασμού μπλοκ εντολών, του **try** και του **catch**, όπως για παράδειγμα φαίνεται στον κώδικα που ακολουθεί.

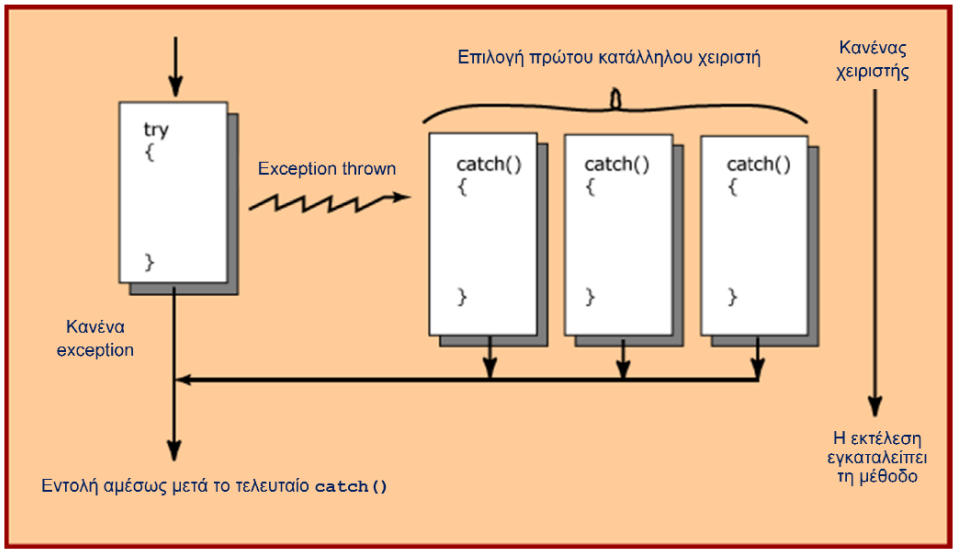
```
try {
    // κώδικας που μπορεί να προκαλέσει exception
}
catch(MyException) {
    // κώδικας που χειρίζεται το exception
}
```

- Ένα **try-catch** μπλοκ είναι ο πιο τυπικός τρόπος χειρισμού εξαιρέσεων και λειτουργεί ως εξής: Ο κώδικας ο οποίος μπορεί να προκαλέσει (raise) μία ή περισσότερες εξαιρέσεις τοποθετείται στο **try** μπλοκ. («επικίνδυνος» κώδικας)

Μηχανισμός χειρισμού εξαιρέσεων (2/4)

- Ακολουθούν ένα ή περισσότερα **catch** μπλοκς, κάθε ένα από τα οποία χειρίζεται ένα είδος εξαίρεσης. Για τον λόγο αυτόν, τα **catch** μπλοκς ονομάζονται και χειριστές (handlers).
- Ένα σωστά δομημένο κομμάτι κώδικα που χειρίζεται έναν αριθμό εξαιρέσεων, θα πρέπει να γραφτεί βάσει κανόνων:
  - Ξεκινάμε από το **try** μπλοκ όπου τοποθετούμε τον κώδικα ο οποίος μπορεί να προκαλέσει μία ή περισσότερες εξαιρέσεις.
  - Στη συνέχεια θα δούμε με ποιον τρόπο γνωρίζουμε ποιες ακριβώς γραμμές μπορεί να προκαλέσουν **exception**.
  - Ο compiler απαιτεί αμέσως μετά από ένα **try** μπλοκ να βρει είτε ένα **catch** μπλοκ είτε ένα **finally** μπλοκ (θα το εξετάσουμε στη συνέχεια). Μάλιστα, δε θα πρέπει να παρεμβάλλεται άλλος κώδικας, δηλαδή τα **catch** /**finally** μπλοκς θα πρέπει να είναι γραμμένα αμέσως μετά το τέλος του try. Αντίστοιχα, αν έχουμε περισσότερα του ενός **catch** μπλοκ πρέπει να είναι γραμμένα το ένα μετά το άλλο χωρίς να παρεμβάλλονται άλλες εντολές.

Μηχανισμός χειρισμού εξαιρέσεων (3/4)



Μηχανισμός χειρισμού εξαιρέσεων (4/4)

- Ο σωστός τρόπος γραφής handlers είναι ακολουθώντας τη σειρά με την οποία μπορούν να προκύψουν εξαιρέσεις, σύμφωνα με τον κώδικα του **try** μπλοκ.
- Τοποθετούμε στην αρχή ειδικότερους χειριστές, δηλαδή αυτούς που στοχεύουν σε συγκεκριμένες εξαιρέσεις, ενώ προχωρώντας προς τα κάτω τοποθετούμε τους γενικότερους, καταλήγοντας στον γενικό χειριστή τον οποίο γράφουμε πάντοτε τελευταίο.
- Το γενικό **catch** μπλοκ (catch-all) μπορεί να «πιάσει» όλες τις εξαιρέσεις που κληρονομούν από την κλάση **Exception** και είναι γνωστές και ως checked exceptions.
- Η σύνταξη του είναι η:

```
catch (Exception e) {
        …
}
```

Το finally-block (1/3)

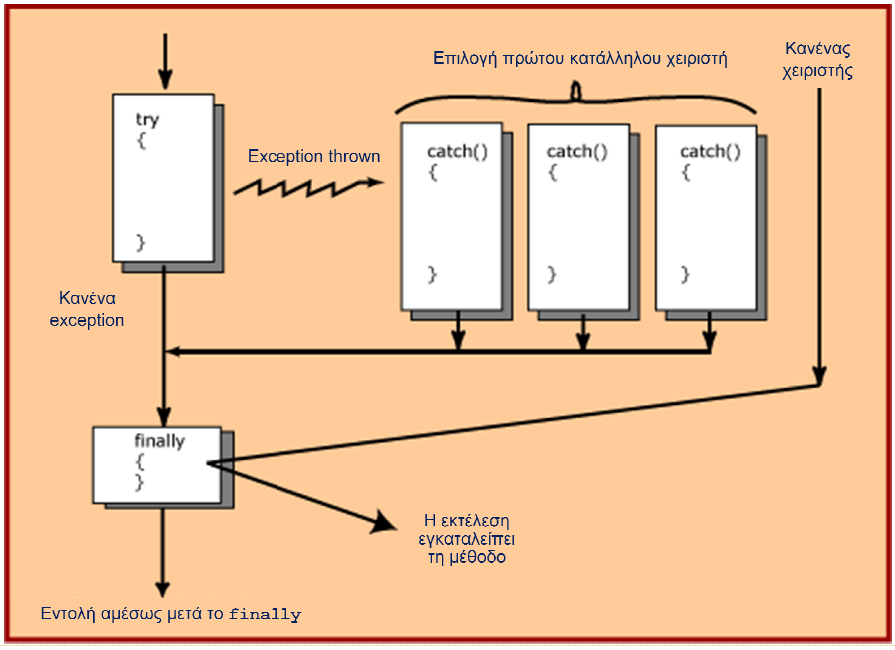
- Το συγκεκριμένο μπλοκ εντάχθηκε στο μηχανισμό για την αντιμετώπιση ανεπιθύμητων καταστάσεων που ο κώδικάς μας έχει δεσμεύσει κάποιους πόρους και κατά την εκτέλεσή του προκύπτει ένα σοβαρό σφάλμα με συνέπεια το πρόγραμμα να τερματίσει χωρίς προηγουμένως να τους αποδεσμεύσει.
- Το σενάριο αυτό μπορεί να αποφευχθεί με τη χρήση ενός **finally** μπλοκ. Πρόκειται για ένα κομμάτι κώδικα που εκτελείται πάντα είτε δηλαδή προκύψει εξαίρεση είτε όχι. Αυτό το κάνει ιδανικό σημείο για την τοποθέτηση κώδικα αποδέσμευσης πόρων.

Το finally-block (2/3)

- Παράδειγμα της βασικής δομής ενός **try-catch-finally** μπλοκ που θα χρησιμοποιούσαμε σε κώδικα που συνδέεται με μία βάση δεδομένων.

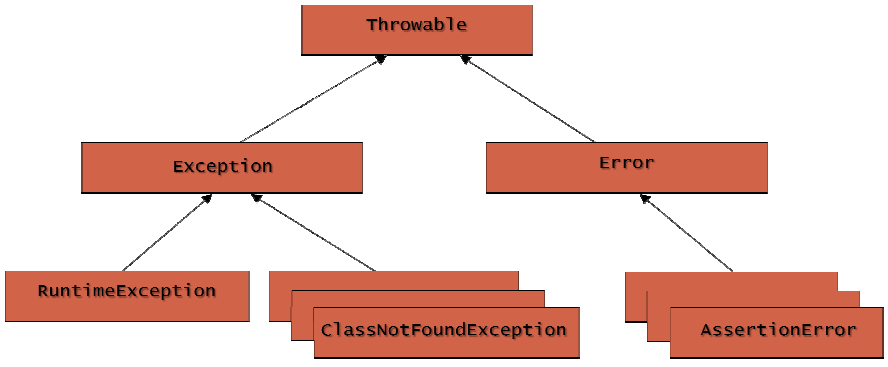
```
try {
   // κώδικας που πραγματοποιεί
  //σύνδεση με μια βάση δεδομένων
}
catch (SQLException e) {
  // κώδικας χειρισμού
}
finally {
  // κώδικας αποσύνδεσης από τη βάση
}
```

To finally-block (3/3)



Τύποι Εξαιρέσεων (1/3)

- Η ιεραρχία κλάσεων της Java που αναπαριστούν τους διάφορους τύπους σφαλμάτων που μπορούν να προκύψουν κατά την εκτέλεση ενός προγράμματος



Τύποι Εξαιρέσεων (2/3)

- Υποκλάσεις της RuntimeException:
  - **ArithmeticException**
  - **ArrayIndexOutOfBoundsException**
  - **ClassCastException**
  - **IllegalArgumentException** & **NumberFormatException**
  - **IllegalStateException**
  - **NullPointerException**

Τύποι Εξαιρέσεων (3/3)

- Υποκλάσεις της Error:
  - **AssertionError**
  - **ExceptionInInitializerError**
  - **IOError**
  - **NoClassDefFoundError**
  - **NoClassDefFoundError**

Συμπεριφορά του catch

```
catch (IOException e) {
        e.printStackTrace();
        System.err.println(e.getMessage());
}
```

Δημιουργία user defined Εξαιρέσεων (1/3)

```
import exception.*;
public class Employee {
        private String fullName;
        private String insuranceBody;
        public Employee(){}
        public Employee(String f, String ib)
                throws InvalidInsuranceBodyException {
                if (!ib.equals("ΙΚΑ") && !ib.equals("ΤΣΜΕΔΕ"))
                        throw new InvalidInsuranceBodyException();
                else {
                        fullName = f;
                        insuranceBody = ib;
                }
        }
```

Δημιουργία user defined Εξαιρέσεων (2/3)

```
public String getFullName() {
        return fullName;
}
public void setFullName(String f) {
        fullName = f;
}
public String getInsuranceBody() {
        return insuranceBody;
}
public void setInsuranceBody(String ib)
        throws InvalidInsuranceBodyException {
                if (!ib.equals("ΙΚΑ") && !ib.equals("ΤΣΜΕΔΕ"))
                        throw new InvalidInsuranceBodyException();
                else
                        insuranceBody = ib;
}
}
```

Δημιουργία user defined Εξαιρέσεων (3/3)

```
package exception.*;
public class InvalidInsuranceBodyException extends Exception {
        private static final long serialVersionUID = 1L;
        // methods
        public InvalidInsuranceBodyException(){
                super("Μη αποδεκτός ασφαλιστικός φορέας");
        }
        public InvalidInsuranceBodyException(String message){
                super(message);
        }
}
```