Διάλεξη 14: Εξαιρέσεις (Exceptions)

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Εξαιρέσεις (Exception)
- Δήλωση, Διαχείριση, Έγερση Εξαιρέσεων
- try, catch, finally, try-with-resources
- Assertions

Διδάσκων: Παναγιώτης Ανδρέου

Εξαίρεση (Exception)

Ερωτήσεις:

- Τι συμβαίνει όταν για κάποιο λογικό λάθος ένα πρόγραμμα παράξει runtime error;
- Πως μπορούμε να διαχειριστούμε αυτό το λάθος ώστε το πρόγραμμα να συνεχίσει την κανονική του ροη;

Απάντηση:

- Με την χρήση των εξαιρέσεων (exceptions)

Ορισμός: "Μία εξαίρεση είναι ένα συμβάν, το οποίο συμβαίνει κατά τη διαρκεί εκτέλεσης ενός προγράμματος και διαταράσσει την κανονική ροή των εντολών του προγράμματος"

exception είναι συντομογραφία της φράσης "**exceptional event**."

Σφάλματα προγραμμάτων

- Τα σφάλματα ενός προγράμματος προκαλούνται από διάφορες καταστάσεις, όπως:
  - Αποτυχία δημιουργίας μνήμης **(new)**
  - array[-1] **(Πρόσβαση σε μη επιτρεπτή θέση μνήμης)**
  - 10/0 **(Μη επιτρεπτές πράξεις με αριθμούς)**
  - void play(char* whatToPlay); play(NULL); **(Λανθασμένη είσοδος παραμέτρων)**
- Κάποια σφάλματα τα προλαμβάνει ο μεταγλωττιστής
- Κάποια άλλα πάλι ΟΧΙ.
- Η γλώσσα προγραμματισμού C και οι περισσότερες γλώσσες παρέχουν μηχανισμούς αντιμετώπισης σφαλμάτων, οι οποίοι βασίζονται περισσότερο σε προγραμματιστικές συμβάσεις και δεν εντάσσονται στη σημασιολογία της γλώσσας.
- Επιστροφή κωδικών λάθους κλπ.

Σφάλματα προγραμματιστών

- Πολύ συχνά οι προγραμματιστές δεν ακολουθούν κάποια σύμβαση αναφορικά με τα σφάλματα. Έτσι, ο κώδικάς τους:
  - Δεν ανιχνεύει σφάλματα.
  - Δεν επιστρέφει τις κατάλληλες τιμές ή δεν εγείρει τις κατάλληλες σημαίες όταν ανιχνεύσει κάποιο σφάλμα.
  - Δεν ελέγχει τις τιμές επιστροφής διαδικασιών ή τυχούσες σημαίες σφαλμάτων.
- Η εισαγωγή κώδικα διαχείρισης σφαλμάτων στα προγράμματα, έχει ως αποτέλεσμα ο κώδικας να γίνεται πολύπλοκος και δυσανάγνωστος.

Προσέγγιση της JAVA

- Η Java εντάσσει τη διαχείριση σφαλμάτων στη σημασιολογία της γλώσσας, επιβάλλοντας την ρητή αντιμετώπισή τους. ΠΩΣ ?
- **Εξαίρεση (Exception):**
  - Προβλήματα τα οποία αποτρέπουν την συνέχιση της μεθόδου ή του πεδίου εμβέλειας στο οποίο βρίσκεται ο κώδικάς μας κατά την εκτέλεσή του.
  - Προβλήματα για τα οποία δεν υπάρχει αρκετή πληροφορία στο τρέχον πεδίο εμβέλειας για να αντιμετωπισθούν και επομένως η κανονική ροή του προγράμματος πρέπει να διακοπεί.
- Όταν σε κάποιο σημείο το σύστημα εκτέλεσης ανιχνεύσει ένα σφάλμα (δηλαδή μια κατάσταση την οποία ο κώδικας στο σημείο αυτό δεν μπορεί να διαχειρισθεί), τότε εγείρει μια εξαίρεση και το πρόβλημα μετατίθεται σε ένα υψηλότερο συγκείμενο, το οποίο ίσως διαθέτει πληροφορίες για την κατάλληλη διαχείριση του σφάλματος.

Τι είναι η εξαίρεση?

- java.lang.Object
  - java.lang.Throwable — **SuperClass**
    - java.lang.Exception
- Κλάσεις που κληρονομούν άμεσα από την java.lang.Exception

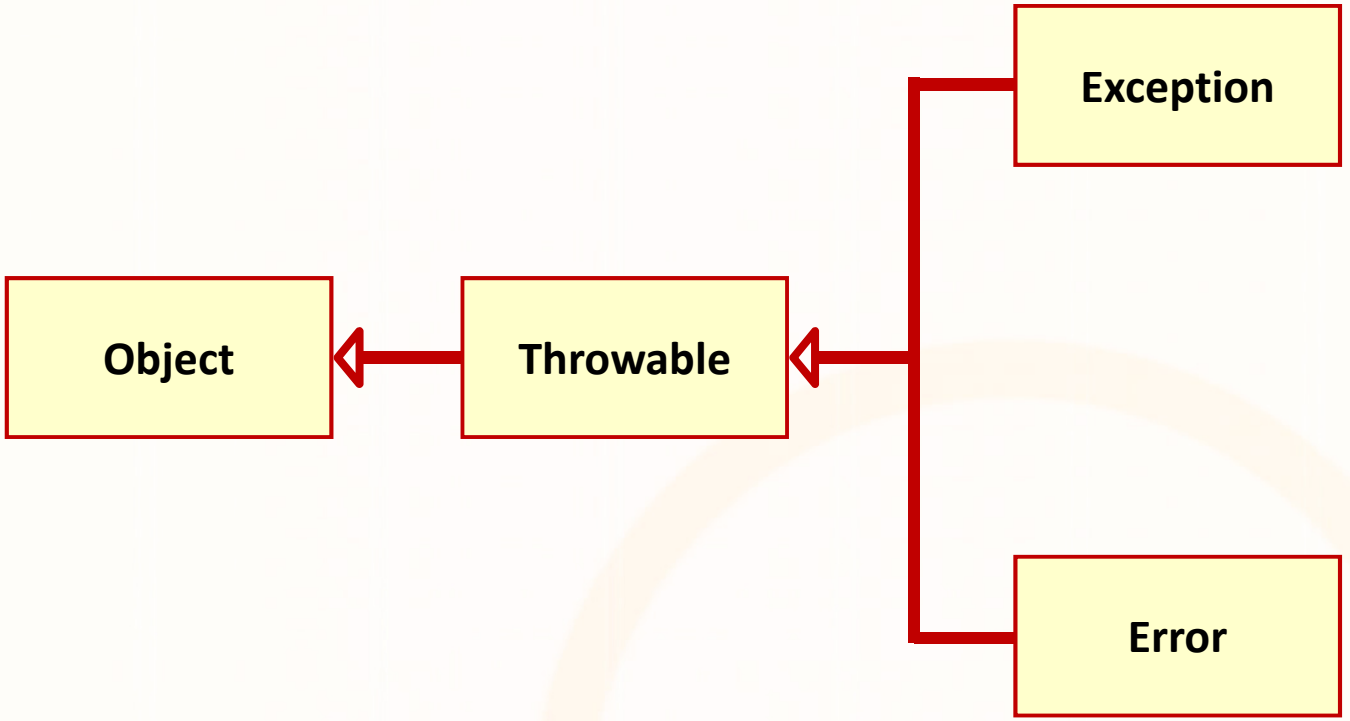
- Στην Java όπως και στην C# κατηγοριοποιούνται σε ιεραρχικές δομές
- Δηλαδή υπάρχουν κλάσεις που κληρονομούν από τους κληρονόμους της Exception.
- Η τεχνική αυτή προσδίδει μία επιπλέον ιδιότητα στη διαχείριση των εξαιρέσεων
  - Αν ένας τομέας κώδικα προστατεύεται για τη διαχείριση της εξαίρεσης Χ τότε προστατεύεται αυτόματα και από τους απόγονους της Χ.

Βασικές εξαιρέσεις στη JAVA

- Η κλάση Throwable περιγράφει οτιδήποτε μπορεί να εγερθεί σαν εξαίρεση, και έχει δύο γενικές υποκλάσεις:
  - **Error:** αντιστοιχεί σε σφάλματα στη διάρκεια της μεταγλώττισης ή σφάλματα συστήματος, για τα οποία ο προγραμματιστής δεν χρειάζεται να τα πιάσει.
  - **Exception:** ο βασικός τύπος σφάλματος ο οποίος μπορεί να εγερθεί από οποιαδήποτε μέθοδο των καθιερωμένων βιβλιοθηκών της Java.
- Ο καλύτερος τρόπος για να δείτε ποιές είναι οι διάφορες εξαιρέσεις που εγείρει η Java, είναι να **πλοηγηθείτε στο εγχειρίδιό της**.
- **Η διαφορά των εξαιρέσεων** μεταξύ τους **έγκειται κατά κύριο λόγο** στα **διαφορετικά ονόματα** που έχουν - μέσω των οποίων σηματοδοτείται το πρόβλημα στο οποίο αντιστοιχούν.
- Οι διάφορες εξαιρέσεις ορίζονται στη βιβλιοθήκη java.lang, αλλά και σε άλλες υποστηρικτικές βιβλιοθήκες όπως util ,net, και io.

Είδη Εξαιρέσεων (Exception Types)

- **Δύο βασικά Είδη Exceptions**
  - Exception
  - Error



Διαπροσωπεία Throwable

Συχνή Χρήση

- String toString(): Returns a short description of this throwable.
- String getMessage(): Returns the detail message string of this throwable.
- StackTraceElement[] getStackTrace(): Provides programmatic access to the stack trace information printed by printStackTrace().
- void printStackTrace(), (PrintStream s), (PrintWriter s): Prints this throwable and its backtrace to the standard error stream.

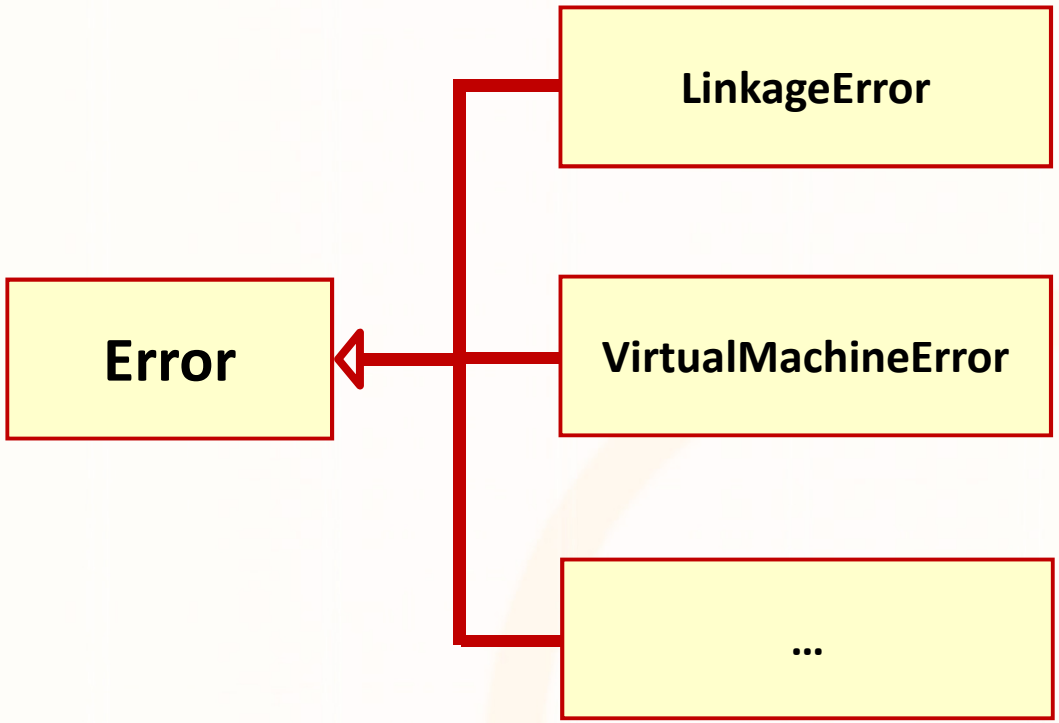
Διαπροσωπεία Throwable

Σπάνια Χρήση

- Throwable fillInStackTrace(): Fills in the execution stack trace.
- Throwable getCause(): Returns the cause of this throwable or null if the cause is nonexistent or unknown.
- String getLocalizedMessage(): Creates a localized description of this throwable.
- Throwable initCause(Throwable cause): Initializes the cause of this throwable to the specified value.
- void setStackTrace(StackTraceElement[] stackTrace): Sets the stack trace elements that will be returned by getStackTrace() and printed by printStackTrace() and related methods.

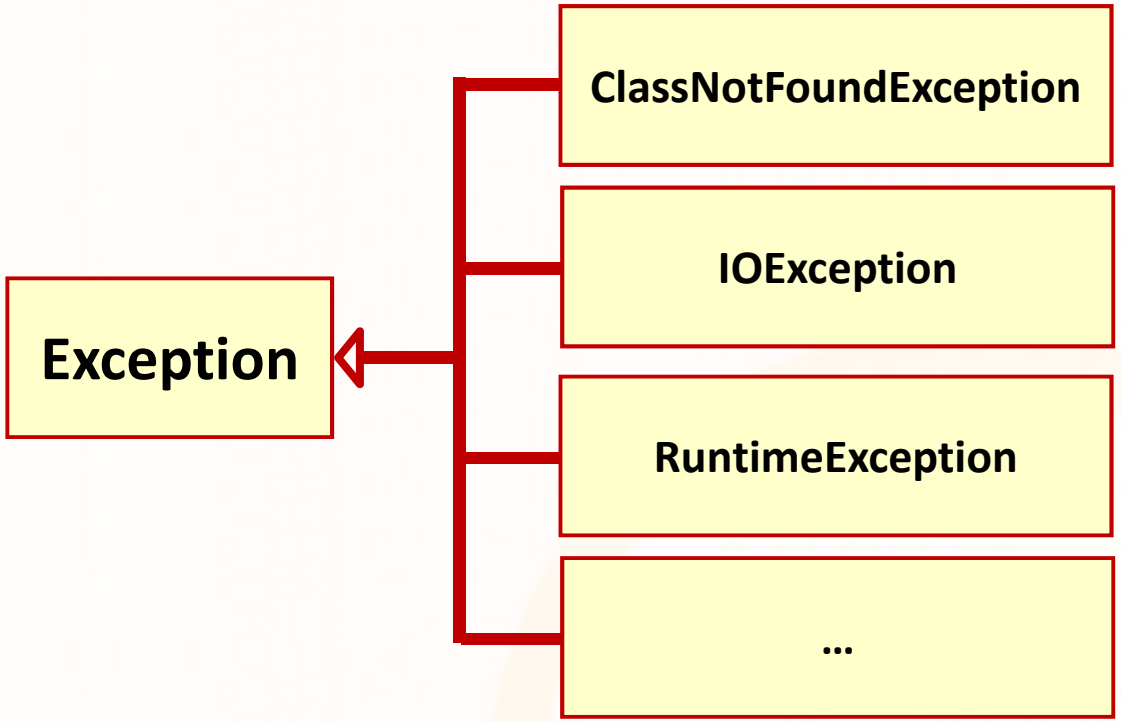
Λάθη Συστήματος (System Errors)

- **Λάθη Συστήματος (system errors):** ρίχνονται από το JVM και αναπαριστούνται από την κλάση Error
- Είναι εσωτερικά λάθη του συστήματος
- Συμβαίνουν σπάνια, αλλά αν συμβεί κάποιο, σαν προγραμματιστές δεν μπορείτε να κάνετε πολλά ☺



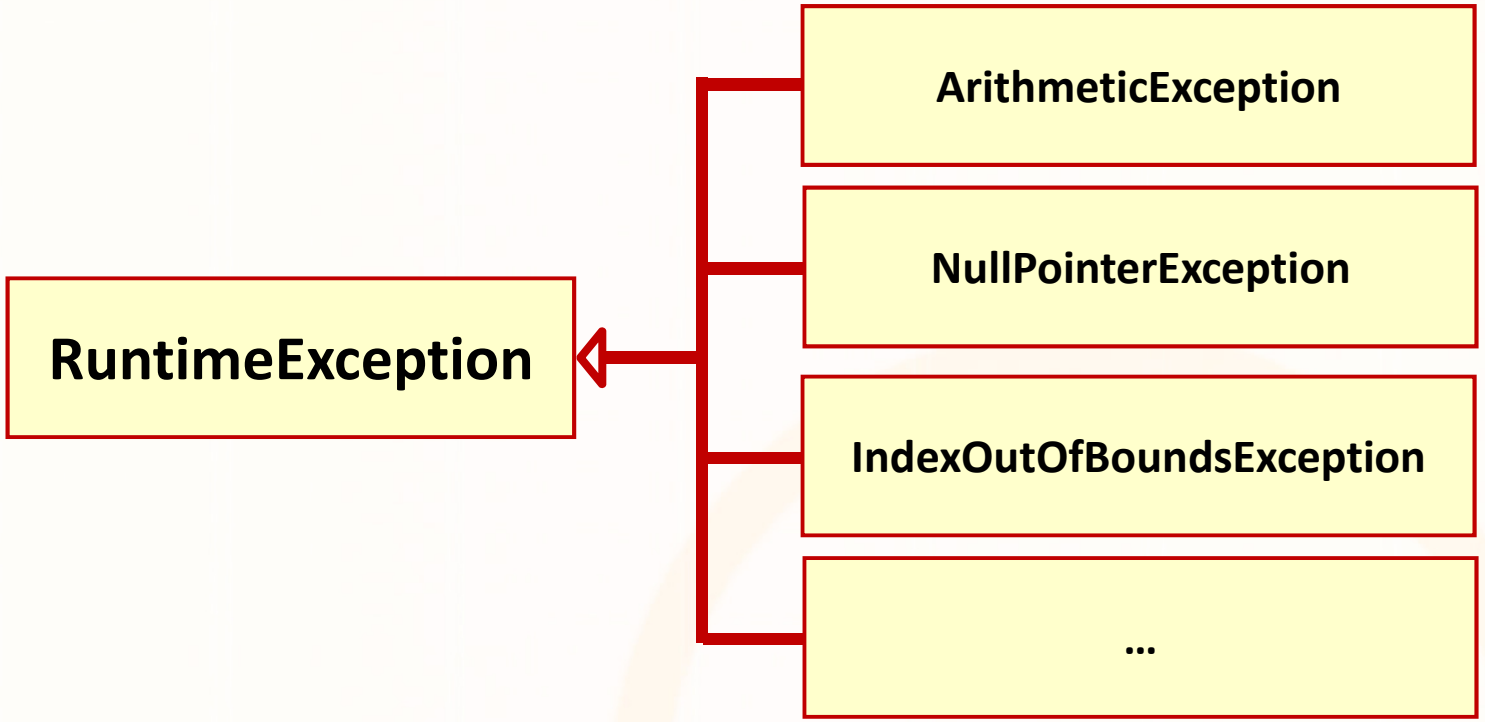
Εξαιρέσεις (Exceptions)

- **Εξαιρέσεις (Exceptions):** αναπαριστούν λάθη τα οποία πηγάζουν από το πρόγραμμα (Runtime Exceptions) και εξωτερικές οντότητες.
- Τα λάθη αυτά μπορούν να διαχειριστούν μέσα από το πρόγραμμα με διάφορους τρόπου όπως try{} catch{}.



Εξαιρέσεις Πραγματικού Χρόνου (Runtime Except.)

- **Εξαιρέσεις Πραγματικού Χρόνου (RuntimeExceptions):** αναπαριστούν λάθη τα οποία πηγάζουν από το πρόγραμμα (Runtime Exceptions).
- Παραδείγματα: λάθος casting, εκτός ορίων, αριθμητικά λάθη, κτλ.

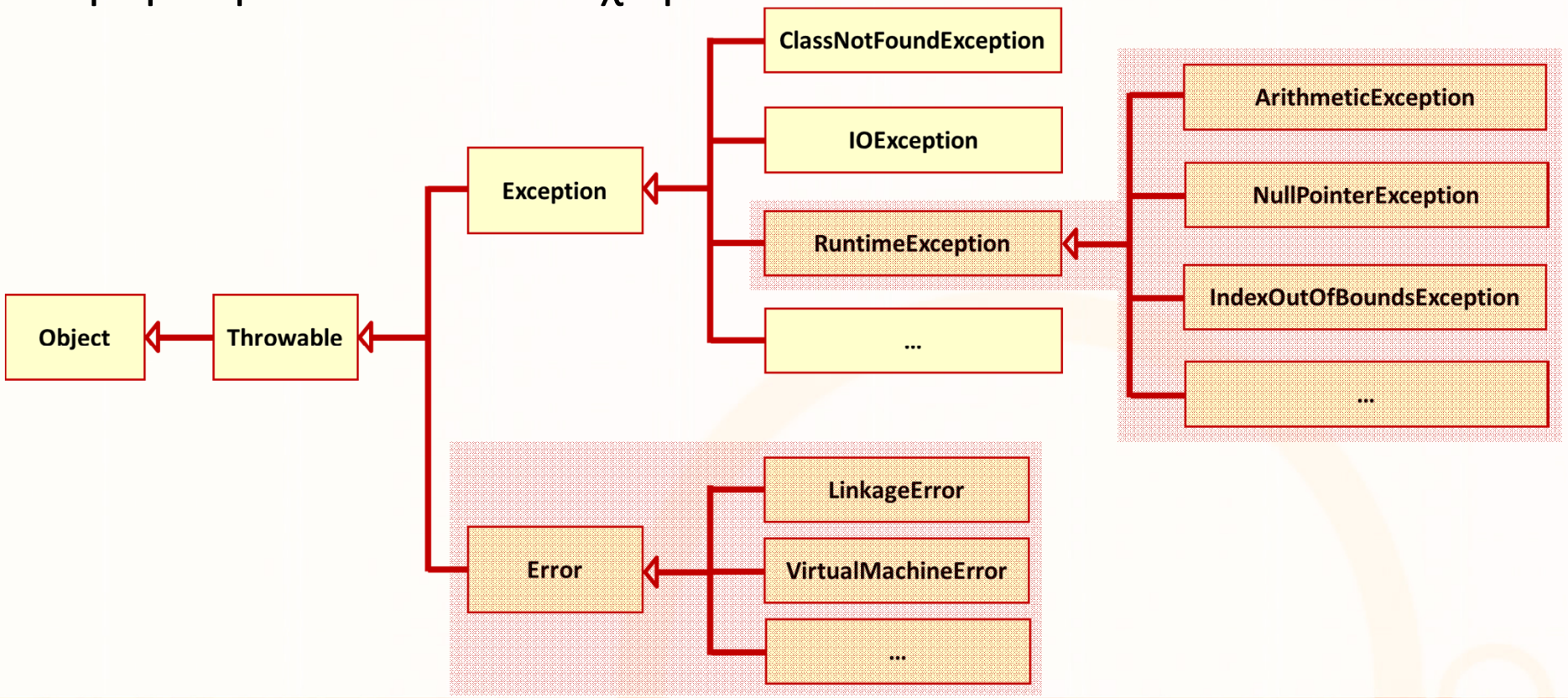


Runtime Exceptions

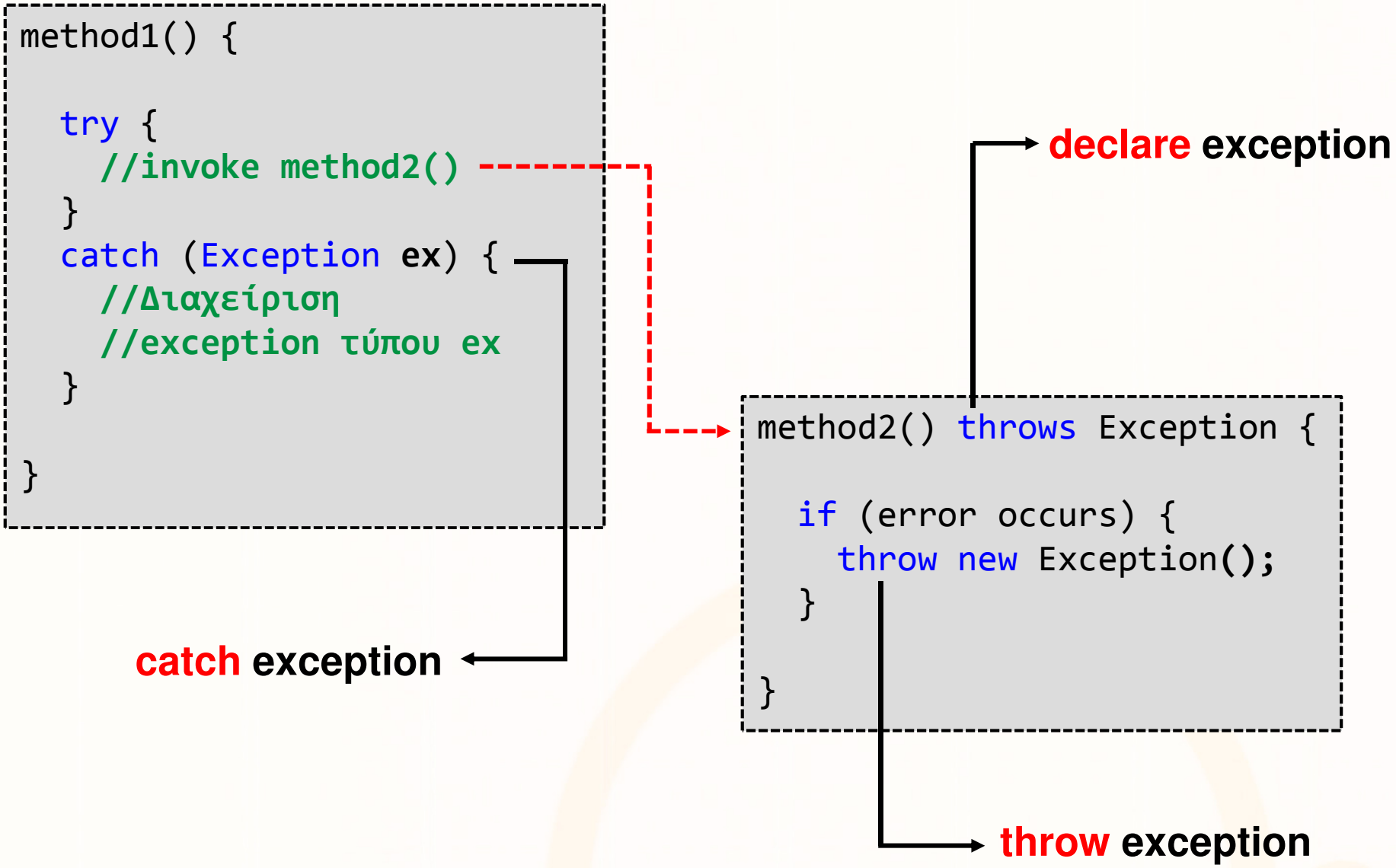
- Στη JAVA υπάρχει μια ολόκληρη **κατηγορία από εξαιρέσεις** οι οποίες εγείρονται αυτόματα από το σύστημα εκτέλεσης της γλώσσας και για **τις οποίες ο προγραμματιστής δεν υποχρεούται από το μεταγλωττιστή** να τις λάβει υπόψη του στον προσδιορισμό των εξαιρέσεων των μεθόδων του.
- Όλες αυτές οι εξαιρέσεις είναι **κληρονόμοι** μιας κλάσης βάσης, **της RuntimeException**. Οι εξαιρέσεις αυτού του τύπου αντιστοιχούν συνήθως είτε σε καταστάσεις τις οποίες δεν μπορεί να ελέγξει ο προγραμματιστής (π.χ. πέρασμα λάθος τιμής παραμέτρου σε μια μέθοδο του) είτε σε λογικά σφάλματα του προγράμματος.
- Εάν μια εξαίρεση τύπου RuntimeException δεν ανιχνευθεί από κάποιο διαχειριστή εξαίρεσης, θα “φθάσει” μέχρι το πλαίσιο συμφραζομένων της main, και θα **προκαλέσει έξοδο του προγράμματος με προηγούμενη κλήση της PrintStackTrace**, για διευκόλυνση της αποσφαλμάτωσης.

Checked vs. Unchecked Exceptions

- **UnChecked Exceptions:** αποτελούν οι εξαιρέσεις των κλάσεων RuntimeException και Error. Συνήθως δεν μπορούν να προβλεφθούν και πολύ απίθανο να διαχειριστούν.
- **Checked Exceptions:** όλα τα άλλα exceptions. Μπορούν να προβλεφθούν και να διαχειριστούν.



Declaring, Throwing, Catching Exceptions



Ο προσδιορισμός των εξαιρέσεων

- Η Java ενθαρρύνει την **ενημέρωση των μεθόδων με τις πιθανές εξαιρέσεις** που μπορούν να εγείρουν και δεν τις διαχειρίζονται.
- Για τον σκοπό αυτό, το συντακτικό της γλώσσας παρέχει την **δήλωση `throws`** με την οποία δηλώνεται ποιές εξαιρέσεις μπορεί να εγερθούν στο σώμα κάποιας μεθόδου **(προσδιορισμός εξαιρέσεων – exception specification)**.
- Η χρήση της throws είναι υποχρεωτική και επιβάλλεται από τον μεταγλωττιστή, ακολουθώντας τη δήλωση των παραμέτρων της μεθόδου:

```
void f() throws TooBig, TooSmall, DivZero { … }
```

- Υπάρχει η δυνατότητα μια μέθοδος να δηλώσει ότι εγείρει κάποια εξαίρεση ενώ δεν περιλαμβάνει στο σώμα της την αντίστοιχη εντολή throw (**γιατί παρέχεται η δυνατότητα αυτή**;)

Declaring Exceptions

- Κάθε μέθοδος πρέπει να δηλώσει ρητά τι είδη exceptions μπορεί να εγείρει (throw)
- Παραδείγματα:
  - `public void method()`
    `throws IOException`
  - `public void method()`
    `throws IOException,` **checked exception**

    `ArrayIndexOutOfBoundsException`
    unchecked exception
- Τα unchecked exceptions δεν είναι ανάγκη να δηλωθούν

Παράδειγμα από JAVA API

public void **connect**(SocketAddress endpoint, int timeout)
throws IOException

- **Description:** … **Parameters: …**
- **Throws:**
  - **IOException** - if an error occurs during the connection
  - **SocketTimeoutException** - if timeout expires before connecting
  - **IllegalBlockingModeException** - if this socket has an associated channel, and the channel is in non-blocking mode
  - **IllegalArgumentException** - if endpoint is null or is a SocketAddress subclass not supported by this socket

Throwing Exceptions

```
if (t == null)  throw new NullPointerException();
if (t == null)  throw new NullPointerException("t==null");
```

- Η εντολή **throw** διακόπτει την τρέχουσα ροή εκτέλεσης και επιστρέφει από την τρέχον εγγραφή δραστηριοποίησης, **μεταφέροντας τον έλεγχο του προγράμματος σε κάποιον διαχειριστή εξαιρέσεων** (η **throw** είναι μία μορφή **return**).
- Με **χρήση** της **throw** μπορούμε να **εγείρουμε οποιοδήποτε αντικείμενο εξαίρεσης**, το οποίο **ανήκει (ή κληρονομεί από)** την κλάση **Throwable**.
- Εφόσον χρησιμοποιούμε την εντολή **throw**, πρέπει να θεωρήσουμε ότι όταν αυτή εκτελεσθεί και εγείρει μια εξαίρεση, είναι ξεκάθαρο ποιός διαχειριστής θα αναλάβει να την αντιμετωπίσει.
- Αυτό επιτυγχάνεται με την **τοποθέτηση των εντολών throw** εντός **φυλασσόμενων περιοχών κώδικα** (guarded regions), οι οποίες καθορίζονται από **την εντολή try**.

Τι γίνεται σε περίπτωση εξαίρεσης;

- Δημιουργείται ένα αντικείμενο εξαίρεσης **(exception object)**
- Το **τρέχον μονοπάτι εκτέλεσης διακόπτεται** και επιστρέφεται το χειριστήριο του αντικείμενου εξαίρεσης από το τρέχον πεδίο εμβέλειας.
- **Ο μηχανισμός διαχείρισης σφαλμάτων** του συστήματος εκτέλεσης της JAVA αναλαμβάνει να εντοπίσει το σημείο από το οποίο μπορεί να **συνεχιστεί η εκτέλεση του προγράμματος**.
- Το σημείο αυτό είναι ο διαχειριστής εξαιρέσεων, ο οποίος αναλαμβάνει να ανανήψει το πρόγραμμα από το πρόβλημα και να συνεχίσει την εκτέλεση.

Catching Exceptions/Διαχειριστές Εξαιρέσεων

- Κάθε φορά που προκαλείται εξαίρεση το πρόγραμμα καλεί τον **διαχειριστή εξαίρεσης (exception handler)**. Έτσι:
  - Περιορίζεται ο κώδικας τον οποίο πρέπει να γράψουμε για ανίχνευση σφαλμάτων.
  - Μπορούμε να διαχωρίσουμε τον κώδικα που περιγράφει το τι θέλουμε να επιτύχουμε από τον κώδικα που διαχειρίζεται τα σφάλματα.

```
try {
   //Κώδικας που μπορεί να
   //δημιουργήσει exceptions
}

catch (Exception ex ) {
   //Διαχείριση/Αντιμετώπιση
   //γενικού τύπου exception
}
```

Η εντολή try και οι διαχειριστές εξαιρέσεων catch

```
try {
   //Κώδικας που μπορεί να
   //δημιουργήσει exceptions
}
catch (ExceptionType1 et1 ) {
   //Διαχείριση/Αντιμετώπιση
   //exception τύπου Type1
}
catch (ExceptionType2 et2 ) {
   //Διαχείριση/Αντιμετώπιση
   //exception τύπου Type2
}
catch (ExceptionType3 et3 ) {
   //Διαχείριση/Αντιμετώπιση
   //exception τύπου Type3
}
catch (Exception ex ) {
   //Διαχείριση/Αντιμετώπιση
   //γενικού τύπου exception
}
```

- Διαχειριστές Εξαιρέσεων (Exception Handlers)
- Οι διαχειριστές τοποθετούνται αμέσως μετά το τέλος της φυλασσόμενης περιοχής.
- Εάν προκληθεί εξαίρεση, ο μηχανισμός διαχείρισής της ψάχνει για τον πρώτο διαχειριστή (catch), η παράμετρος του οποίου συμπίπτει με τον τύπο της εξαίρεσης.

Διαχείριση εξαιρέσεων

- Στα προγράμματά μας μπορούμε να εισαγάγουμε διαχειριστές εξαιρέσεων, οι οποίοι να διαχειρίζονται οποιεσδήποτε εξαιρέσεις. Αυτό το επιτυγχάνουμε πιάνοντας **εξαιρέσεις (γενικού) τύπου Exception**.
- Επειδή η Exception δεν παρέχει πολλές πληροφορίες για το συγκείμενο μιας εξαίρεσης, έχουμε τη δυνατότητα να **ορίσουμε νέες κλάσεις εξαιρέσεων, κληρονομώντας από την Exception** και παρακάμπτοντας τις μεθόδους της Exception, της Throwable και της Object:
  - `getMessage()`
  - `getLocalizedMessage()`
  - `toString()`
  - `printStackTrace()`
  - `fillInStackTrace()`
  - `getClass()`

Η εντολή finally

```
try {
    //Κώδικας που μπορεί να
    //δημιουργήσει exceptions
}
catch (ExceptionType1 et1 ) {
    //Διαχείριση/Αντιμετώπιση
    //exception τύπου Type1
}
…
catch (Exception ex ) {
    //Διαχείριση/Αντιμετώπιση
    //γενικού τύπου exception
}

finally {
    //Εκτελείτε πάντα
}
```

- **Το finally block εκτελείται πάντα!**
- Εξασφαλίζει ότι κάποιες δηλώσει θα εκτελεστούν άσχετα αν συμβεί κάποιο απρόσμενο exception
- Μπορεί να χρησιμοποιηθεί και χωρίς δηλώσεις
- Επιτρέπει την εκτέλεση κώδικα ακόμα και στις περιπτώσεις που εκτελείτε κατά λάθος κάποιο return, continue ή break
- Πολύ καλή πρακτική

Η εντολή try-with-resources

```
try ( BufferedReader br =
        new BufferedReader(
         new FileReader(path))) {
   //Κώδικας που μπορεί να
   //δημιουργήσει exceptions
}
…
```

- Παρόμοιο με τη δήλωση try αλλά **επιτρέπει και τη αρχικοποίηση κάποιων πόρων (resources)**
- Ένας **πόρος (resource)** είναι ένα αντικείμενο το οποίο υπάρχει μόνο κατά την εκτέλεση του code block
- Μόνο τα αντικείμενα που υλοποιούν τη διαπροσωπεία java.lang.AutoCloseable μπορούν να χρησιμοποιηθούν σαν resource (περιλαμβάνει όλα τα αντικείμενα που υλοποιούν java.io.Closeable)

Rethrowing Exceptions

- Σε ορισμένες περιπτώσεις επιθυμούμε να εγείρουμε μια εξαίρεση που έχουμε συλλάβει σε κάποιο σημείο του κώδικά μας, ώστε η διαχείριση της να αναληφθεί από το αμέσως επόμενο συγκείμενο.
- Αυτό προκύπτει αν “πιάσουμε” μια γενική εξαίρεση (Exception) και θελήσουμε να την παραπέμψουμε σε έναν πιο εξειδικευμένο διαχειριστή.
  - Σε τέτοια περίπτωση, το αντικείμενο της εξαίρεσης διατηρείται και μεταφέρεται στον επόμενο διαχειριστή, ο οποίος έχει πρόσβαση στα περιεχόμενα του αντικειμένου εξαίρεσης.

Rethrowing Exceptions

- Όταν **προωθούμε μια εξαίρεση**, τα περιεχόμενα της **printStackTrace() εξακολουθούν να αφορούν στο σημείο στο οποίο προκλήθηκε η αρχική εξαίρεση**, και **όχι** στο σημείο στο οποίο επανεγείραμε την εξαίρεση.
- Αν θέλουμε να συνδυάσουμε το αντικείμενο εξαίρεσης με **νέα πληροφορία** για το **ίχνος της στοίβας** (stack trace), μπορούμε να καλέσουμε τη μέθοδο **fillInStackTrace().**
- Η **fillInStackTrace()** δημιουργεί ένα αντικείμενο Throwable, προσθέτοντας την τρέχουσα κατάσταση της στοίβας στο αρχικό αντικείμενο της εξαίρεσης.
- Το νέο αντικείμενο Throwable, επιστρέφεται από την fillInStackTrace().
- Αν προωθήσουμε εξαίρεση διαφορετικού τύπου, τότε χάνουμε την πληροφορία σχετικά με το σημείο όπου εγέρθηκε η αρχική εξαίρεση και λαμβάνουμε (μέσω της printStackTrace) πληροφορίες για το σημείο της νέας εντολής throw.

Rethrowing Exceptions

```
public class Rethrowing {

   public static void f() throws Exception {
      System.out.println("1. originating the exception in f()");
      throw new Exception(“***thrown from f()");}

   public static void g() throws Throwable {
      try { f(); } catch(Exception e) {
         System.err.println(“2.Inside g(), e.printStackTrace()");
         e.printStackTrace(); //3
         throw e;
      }
   }

   public static void main(String[] args) throws Throwable {
      try { g(); } catch(Exception e) {
         System.err.println("4.Caught in main,e.printStackTrace()");
         e.printStackTrace(); //5
      }
   }
}
```

Rethrowing Exceptions

```
$ java Rethrowing
1. originating the exception in f()
2. Inside g(),e.printStackTrace()
3. java.lang.Exception: ***thrown from f()
      at Rethrowing.f(Rethrowing.java:4)
      at Rethrowing.g(Rethrowing.java:8)
      at Rethrowing.main(Rethrowing.java:16)

4. Caught in main, e.printStackTrace()
5. java.lang.Exception: ***thrown from f()
   at Rethrowing.f(Rethrowing.java:4)
   at Rethrowing.g(Rethrowing.java:8)
   at Rethrowing.main(Rethrowing.java:16)
```

Αλυσιδωτές εξαιρέσεις (exception chaining)

- Συχνά θέλουμε, πιάνοντας μια εξαίρεση, να εγείρουμε κάποια άλλη, διατηρώντας ταυτόχρονα την πληροφορία για την αρχική εξαίρεση.
- Από το JDK 1.4 και μετά, οι υποκλάσεις της Throwable δέχονται ένα αντικείμενο cause στον κατασκευαστή τους, το οποίο αποτελεί την αρχική εξαίρεση.
- Περνώντας το cause στον κατασκευαστή, διατηρείται το πλήρες ίχνος της στοίβας μέχρι την αρχική εξαίρεση, παρ' ότι δημιουργούμε και εγείρουμε νέα εξαίρεση.
- Οι υποκλάσεις της Throwable που δέχονται την παράμετρο cause στον κατασκευαστή τους είναι οι θεμελιώδεις εξαιρέσεις Error (used by the JVM to report system errors), Exception , και RuntimeException.
- Αν θέλετε να αλυσοδέσετε άλλους τύπους εξαιρέσεων, πρέπει να το κάνετε μέσω της μεθόδου initCause( ).

Ιεραρχία εξαιρέσεων: Sub-classing

- Υπάρχει η δυνατότητα δημιουργίας δικών μας Εξαιρέσεων κάνοντας χρήση της κληρονομικότητας.
- Παράδειγμα:

```
public class HierarchicalTryCatch
                                extends Exception {
   …
   public HierarchicalTryCatch() {

   }

   public HierarchicalTryCatch(String str) {
      super("HierarchicalTryCatch" + str);
   }
   …
}
```

Ιεραρχία εξαιρέσεων: usage

- Παράδειγμα χρήσης:

```
public class TestExceptions {

   public static void main(String args) {

      try {
         throw new HierarchicalTryCatch("test");
      }
      catch (HierarchicalTryCatch ex1 ) {
         //Διαχείριση/Αντιμετώπιση
         //exception τύπου HierarchicalTryCatch
      }
   }
}
```

Ιεραρχία εξαιρέσεων: usage

- Παράδειγμα χρήσης (καλύτερη πρακτική):

```
public class TestExceptions {

   public static void main(String args) {

      try {
         throw new HierarchicalTryCatch("test");
      }
      catch (HierarchicalTryCatch ex1 ) {
         //Διαχείριση/Αντιμετώπιση
         //exception τύπου HierarchicalTryCatch
      }
      catch (Exception ex ) {
         //Διαχείριση/Αντιμετώπιση
         //γενικού τύπου exception
      }
   }
}
```

Κατασκευή εξαιρέσεων

- Local Parameters, override

```
public class MyException extends Exception {

   private int x;

   public class MyException() { }

   public class MyException(String msg) {
      super(msg);
   }

   public class MyException(String msg, int x) {
      super(msg);
      this.x = x;
   }

   public int getX() {return x;}

   public String getMessage() {
      return “Custom MyException Message”;
   }
}
```

Κατασκευή εξαιρέσεων

- Local Parameters, override

```
public class MyException extends Exception {

   private int x;

   public class MyException() { }

   public class MyException(String msg) {
      super(msg);
   }

   public class MyException(String msg, int x) {
      super(msg);
      this.x = x;
   }

   public int getX() {return x;}

   public String getMessage() {
      return “Custom MyException Message”;
   }
}
```

Περιορισμοί εξαιρέσεων

- Όταν υπερσκελίζουμε (overriding) μια μέθοδο (κατά την εφαρμογή κληρονομικότητας), μπορούμε να εγείρουμε μόνο εξαιρέσεις οι οποίες καθορίζονται στην κλάση-βάσης της μεθόδου που υπερσκελίζουμε.
- Ο περιορισμός αυτός υπάρχει ούτως ώστε κώδικες που δουλεύουν με μια κλάση, να δουλεύουν σωστά και με κάθε αντικείμενο που κληρονομεί από αυτή την κλάση.
- Οι περιορισμοί κληρονομικότητας των εξαιρέσεων δεν ισχύουν για τους κατασκευαστές. Ένας κατασκευαστής μπορεί να εγείρει ότι εξαιρέσεις θέλει.
  - ωστόσο, επειδή η κλήση του εμπεριέχει κλήση στον κατασκευαστή της βάσης του, ο κατασκευαστής πρέπει να δηλώνει όλες τις εξαιρέσεις του κατασκευαστή βάσης του, στον προσδιορισμό των δικών του εξαιρέσεων.
- Σημειώστε ότι ένας κατασκευαστής δεν μπορεί να ανιχνεύει τις εξαιρέσεις που εγείρει ο κατασκευαστής της κλάσης του πατέρα του.

Περιορισμοί εξαιρέσεων

```
class class BaseballException extends Exception {}
class Foul extends BaseballException {}
class Strike extends BaseballException {}

class Inning {
      public Inning() throws BaseballException {}

      public void event() throws BaseballException {}
      public void walk() {}

}

class StormException extends Exception {}
class RainedOut extends StormException {}
class PopFoul extends Foul {}
```

Περιορισμοί εξαιρέσεων

```
public class StormyInning extends Inning {
  // OK to add new exceptions for constructors, but you
  // must deal with the base constructor exceptions:
  public StormyInning() throws RainedOut, BaseballException {}
  public StormyInning(String s) throws Foul, BaseballException {}
  // Regular methods must conform to base class:
  //! public void walk() throws PopFoul {}
  //! public void event() throws RainedOut {}
  // Compile error: Interface CANNOT add exceptions to existing
  // methods from the base class:
  // If the method doesn't already exist in the
  // base class, the exception is OK:
  public void rainHard() throws RainedOut {}
  // You can choose to not throw any exceptions,
  // even if the base version does:
  public void event() {}
}
```

Παρατηρήσεις

- Δεν σημαίνει ότι επειδή μία προδιαγραφή exception υπάρχει στην μέθοδο μίας base-class ότι θα πρέπει να υπάρχει και στην μέθοδο που ορίζεται στην υποκλάση.
- Αυτό είναι διαφορετικό από τους κανόνες κληρονομικότητας που καθορίζουν ότι μία μέθοδος στο base class πρέπει να υπάρχει και στην υποκλάση.
- Με απλά λόγια, οι προδιαγραφές διαπροσωπείας μίας εξαίρεσης για κάποια συγκεκριμένη μέθοδο μπορούν να "στενέψουν" κατά τη διάρκεια της κληρονομικότητας και επικάλυψης αλλά όχι να διευρυνθούν
- Αυτό καθιστά τον αντίθετο κανόνα σε σχέση με τις διαπροσωπείες κατά τη διάρκεια της κληρονομικότητας.

Assertions

- **Assertion: Δήλωση που επιτρέπει την επιβεβαίωση κάποιων υποθέσεων για το πρόγραμμα που εκτελείται.**
- Ισχύουν από JAVA 1.4 και πάνω.
- Περιλαμβάνουν boolean εκφράσεις που πρέπει να είναι true κατά τη διάρκεια εκτέλεσης
- Δηλώνονται με τη λέξη **assert**.
  - assert <έκφραση> π.χ., assert i==10;
  - assert <έκφραση>: <μήνυμα> π.χ., assert (i==10): “i is” + i;
- **Αν το assertion είναι false τότε** θα γίνει throw **AssertionError**
- Η κλάση AssertionError class έχει τον no-arg constructor και 7 overloaded single-argument constructors με τύπο παραμέτρων int, long, float, double, boolean, char, and Object.
- Για το πρώτο παράδειγμα εκτελείται ο no-arg constructor
- Για άλλες περιπτώσεις εκτελείται ο ανάλογος constructor

Παράδειγμα Assertion

```
public class AssertionDemo {
  public static void main(String[] args) {
    int i; int sum = 0;
    for (i = 0; i < 10; i++) {
      sum += i;
    }
    assert i == 10;
    assert (sum > 10 && sum < 5 * 10) :
                      "sum is " + sum;
  }
}
```

Εκτελώντας προγράμματα με assertions

- Τα assertions δεν είναι ενεργοποιημένα by default
- Για να ενεργοποιηθούν πρέπει να εκτελεστεί το switch **–enableassertions**, ή τη συντομογραφία **–ea**

- Μπορούν να ενεργοποιηθούν σε επίπεδο πακέτου ή κλάσης
- Παράδειγμα: **java –ea:package1 –da:Class1 AssertDemo**
- Ενεργοποιεί τα assertions στο πακέτο package1 και τα απενεργοποιεί στην κλάση Class1

Assertions vs. Exceptions

- **Τα assertions δεν πρέπει να αντικαταστούν το exception handling**
- Τα assertions επιβεβαιώνουν την ορθότητα του προγράμματος
- Τα exceptions διαχειρίζονται σφάλματα ➔ ευρωστία
- Τα assertions μπορούν να απενεργοποιηθούν ενώ το exception handling όχι