ΕΞΑΙΡΕΣΕΙΣ στη JAVA

Σφάλματα προγραμμάτων

ΕΠΛ233

- Τα σφάλματα ενός προγράμματος προκαλούνται από διάφορες καταστάσεις, όπως: Εξάντληση διαθέσιμης ιδεατής μνήμης, προσπάθεια ανάγνωσης εκτός των ορίων ενός πίνακα, διαίρεση με το μηδέν, πέρασμα άκυρων παραμέτρων σε μεθόδους, κλπ.
- Ιδανικός χρόνος ανίχνευσης σφαλμάτων:ο χρόνος μεταγλώττισης (αυτό δεν είναι δυνατό για όλα τα σφάλματα).
- Η C και οι περισσότερες γλώσσες παρέχουν μηχανισμούς αντιμετώπισης σφαλμάτων, οι οποίοι βασίζονται περισσότερο σε προγραμματιστικές συμβάσεις και δεν εντάσσονται στη σημασιολογία της γλώσσας. Π.χ.:
  - Σε περίπτωση σφάλματος το οποίο ανιχνεύεται κατά τη στιγμή της εκτέλεσης μιας διαδικασίας, ο προγραμματιστής πρέπει να φροντίσει ώστε η διαδικασία να επιστρέφει κάποια ειδική τιμή, ή να εγείρει κάποια σημαία, ώστε να σηματοδοτήσει ότι κάτι έχει πάει λάθος.
  - Έτσι, διαδικασίες οι οποίες ολοκληρώνονται κανονικά επιστρέφουν τιμή 0, ενώ διαδικασίες στις οποίες ανιχνεύεται κάποιο σφάλμα επιστρέφουν τιμή διαφορετική του μηδενός.
  - Ο κώδικας από τον οποίο καλείται η διαδικασία που εγείρει το σφάλμα, μπορεί να ανιχνεύσει την τιμή επιστροφής της διαδικασίας και να διαχειριστεί αναλόγως την περίπτωση.

Μειονεκτήματα

ΕΠΛ233

- Πολύ συχνά οι προγραμματιστές δεν ακολουθούν οποιαδήποτε 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 δυσκολο-ανάγνωστος.

Προσέγγιση της JAVA

ΕΠΛ233

- Εντάσσει τη διαχείριση σφαλμάτων στη σημασιολογία της γλώσσας, επιβάλλοντας την ρητή αντιμετώπισή τους.
- Για τη διαχείριση σφαλμάτων, η JAVA εισάγει την έννοια της "εξαίρεσης" (exception). Οι εξαιρέσεις (ή συνθήκες εξαιρέσεως) είναι:
  - Πρόβληματα 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.

Εξαιρέσεις στη JAVA

- Η ανίχνευση κάποιου συγκεκριμένου σφάλματος και η διαχείρισή του δεν γίνεται με εισαγωγή κώδικα στα σημεία όπου αυτό μπορεί να προκύψει, αλλά σε ένα μόνο σημείο του κώδικα, τον αποκαλούμενο 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.
- Τι γίνεται σε περίπτωση εξαίρεσης;
  - Δημιουργείται ένα αντικείμενο εξαίρεσης (exception object) - όπως όλα τα αντικείμενα σώζεται στο σωρό.
  - 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 βρεί 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.

Εξαιρέσεις στη JAVA

ΕΠΛ233

```
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 το τρέχον εγγράφημα δραστηριοποίησης, μεταφέροντας τον έλεγχο του προγράμματος σε κάποιον διαχειριστή εξαιρέσεων [η throw είναι ένα είδος 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.

Η εντολή try και οι διαχειριστές εξαιρέσεων

ΕΠΛ233

```
Try {
    // code that might generate exceptions
} catch(Type1 id1) {
    // handle exceptions of type 1
} catch(Type2 id2) {
    // handle exceptions of type 2
} catch(Type3 id3) {
    // handle exceptions of type 3
}
```

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

Κατασκευή εξαιρέσεων

ΕΠΛ233

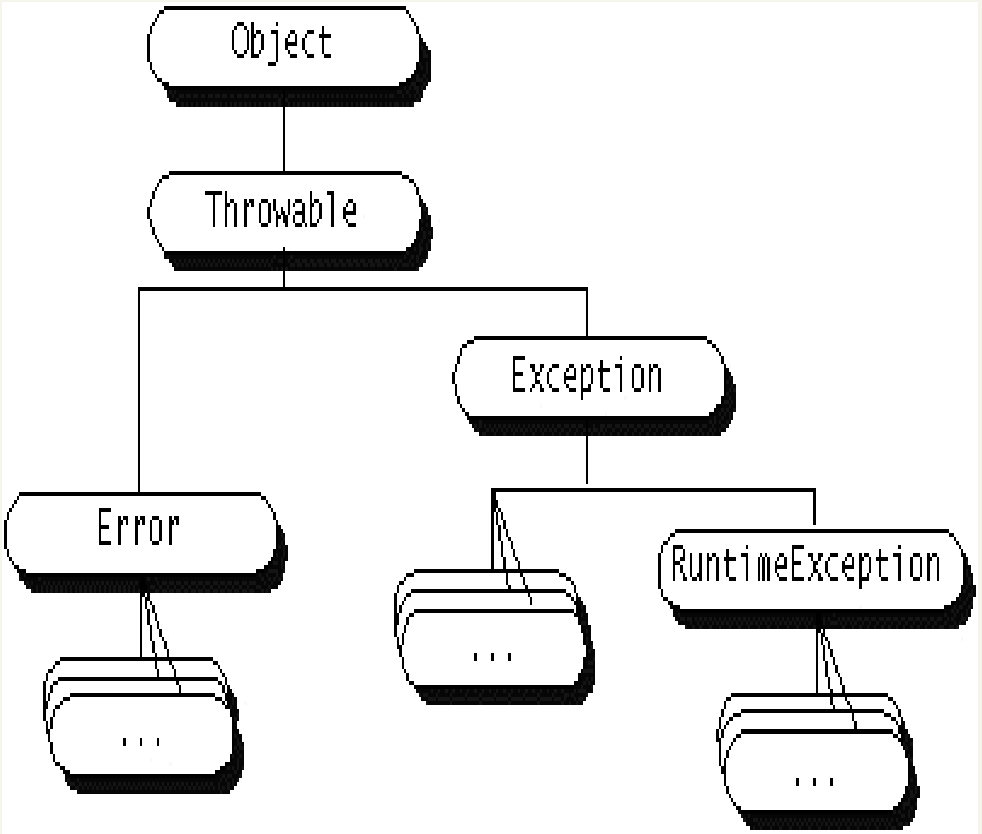
```
class SimpleException extends Exception {}
public class SimpleExceptionDemo {
  public void f() throws SimpleException {
    System.out.println("Throw SimpleException from f()");
    throw new SimpleException();
}
public static void main(String[] args {
  SimpleExceptionDemo sed = new SimpleExceptionDemo();
  try {
     sed.f();
  } catch (SimpleException e) {
     System.err.println("Caught it!");}
}
```

Κατασκευή εξαιρέσεων

ΕΠΛ233

```
class MyException extends Exception {
   public MyException() {}
   public MyException(String msg) { super (msg); }
}
public class FullConstructors {
   public static void f() throws MyException {
    System.out.println( "Throwing MyException from f()" );
       throw new MyException();
   }
   public static void g() throws MyException {
       System.out.println( "Throwing MyException from g()" );
       throw new MyException( "Originated in g()" );
   }
   public static void main(String[] args) {
       try {       f();     }
              catch (MyException e) {e.printStackTrace();}
       try {       g();     }
              catch (MyException e) {e.printStackTrace();}
```

Ιεραρχία κλάσεων εξαιρέσεων



- Οι κλάση Exception της JAVA κληρονομεί από την κλάση Throwable, η οποία έχει μεθόδους όπως:
  - printStackTrace: παρέχει πληροφορίες για την σειρά των μεθόδων οι οποίες εκλήθησαν ώστε να φθάσει η ροή εκτέλεσης στην εντολή η οποία προκάλεσε την εξαίρεση.
  - getMessage: καθορίζει το περιεχόμενο του μηνύματος που εκτυπώνεται στη διάρκεια μιας εξαίρεσης.

Κατασκευή εξαιρέσεων

ΕΠΛ233

```
class MyException2 extends Exception {
   private int x;
   public MyException2() {}
   public MyException2(String msg) {
       super (msg);
   }
   public MyException2(String msg, int x) {
       super(msg);
       this.x = x;
   }
   public int val() {
       return x;
   }
   public String getMessage() {
       return "Detail Message: "+x+" "+super.getMessage();
}}
```

Κατασκευή εξαιρέσεων

ΕΠΛ233

```
public class ExtraFeatures {
   public static void f() throws MyException2 {
       System.out.println("Throwing MyException2 from f()");
       throw new MyException2();}
   public static void g() throws MyException2 {
     System.out.println("Throwing MyException2 from g()");
     throw new MyException2("Originated in g()");}

   public static void h() throws MyException2 {
       System.out.println("Throwing MyException2 from h()");
       throw new MyException2( "Originated in h()" , 47);}

   public static void main(String[] args) {
       try {f();} catch (MyException2 e) {e.printStackTrace();}
       try {g();} catch (MyException2 e) {e.printStackTrace();}
       try {h();} catch (MyException2 e) {
             e.printStackTrace();
             System.err.println( "e.val() = " + e.val());}}
```

Ο προσδιορισμός των εξαιρέσεων

ΕΠΛ233

- Η JAVA ενθαρρύνει την ενημέρωση αυτών που χρησιμοποιούν κάποια μέθοδο σχετικά με τις εξαιρέσεις τις οποίες μπορεί να εγείρει αυτή η μέθοδος (γιατί;).
- Για τον σκοπό αυτό, το συντακτικό της γλώσσας παρέχει την εντολή **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;)

Διαχείριση εξαιρέσεων

ΕΠΛ233

- 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 αναιρώντας μεθόδους της Exception, της Throwable και της Object:
  - getMessage()
  - getLocalizedMessage()
  - toString()
  - printStackTrace()
  - fillInStackTrace()
  - getClass()

Επανέγερση εξαίρεσης

ΕΠΛ233

- 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 πιό εξειδικευμένο διαχειριστή.
  - Σε τέτοια περίπτωση, το αντικείμενο της εξαίρεσης διατηρείται και μεταφέρεται στον επόμενο διαχειριστή, ο οποίος έχει πρόσβαση στα περιεχόμενα του αντικειμένου εξαίρεσης.

Επανέγερση εξαίρεσης

ΕΠΛ233

- Όταν επανεγείρουμε 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().

Επανέγερση εξαίρεσης

ΕΠΛ233

```
public class Rethrowing {
  public static void f() throws Exception {
    System.out.println("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("Inside g(),e.printStackTrace()");
      e.printStackTrace();
      throw e;
    }
  }
  public static void main(String[] args) throws Throwable {
    try { g(); } catch(Exception e) {
      System.err.println("Caught in main, e.printStackTrace()");
      e.printStackTrace();
    } }
}
```

Επανέγερση εξαίρεσης

ΕΠΛ233

```
public class Rethrowing {

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

```
magnaura.cs.ucy.ac.cy>java Rethrowing
originating the exception in f()
Inside g(),e.printStackTrace()
java.lang.Exception: thrown from f()
    at Rethrowing.f(Rethrowing.java:5)
    at Rethrowing.g(Rethrowing.java:10)
    at Rethrowing.main(Rethrowing.java:21)
Caught in main, e.printStackTrace()
java.lang.Exception: thrown from f()
    at Rethrowing.f(Rethrowing.java:5)
    at Rethrowing.g(Rethrowing.java:10)
    at Rethrowing.main(Rethrowing.java:21)
```

Επανέγερση εξαίρεσης

ΕΠΛ233

```
public class Rethrowing {
  public static void f() throws Exception {
    System.out.println("originating exception in f()");
    throw new Exception("thrown from f()");
  }
  public static void g() throws Throwable {
    try { f();  } catch(Exception e) {
      System.err.println("In g(),e.printStackTrace()");
      e.printStackTrace();
      throw e.fillInStackTrace(); // returns Throwable object
    }
  }
  public static void main(String[] args) throws Throwable {
    try { g(); } catch(Exception e) {
      System.err.println("Caught in main, e.printStackTrace()");
      e.printStackTrace();
    } }
}
```

Επανέγερση εξαίρεσης

ΕΠΛ233

```
public class Rethrowing {


  public static void f() throws Exception {
    System.out.println("originating exception in f()");
    throw new Exception("thrown from f()");
  }
  public static void g() throws Throwable {
    try { f();  } catch(Exception e) {
      System.err.println("In g(),e.printStackTrace()");
      e.printStackTrace();
      throw e.fillInStackTrace(); // returns Throwable object
    }
  }
  public static void main(String[] args) throws Throwable {
    try { g(); } catch(Exception e) {
      System.err.println("Caught in main, e.printStackTrace()");
      e.printStackTrace();
    } }
}
```

```
magnaura.cs.ucy.ac.cy>java Rethrowing
originating exception in f()
In g(),e.printStackTrace()
java.lang.Exception: thrown from f()
    at Rethrowing.f(Rethrowing.java:5)
    at Rethrowing.g(Rethrowing.java:10)
    at Rethrowing.main(Rethrowing.java:21)
Caught in main, e.printStackTrace()
java.lang.Exception: thrown from f()
    at Rethrowing.g(Rethrowing.java:15)
    at Rethrowing.main(Rethrowing.java:21)
```

Επανέγερση εξαίρεσης

ΕΠΛ233

- Αν επανεγείρουμε 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.

Επανέγερση εξαίρεσης

ΕΠΛ233

```
class OneException extends Exception {
  public OneException(String s) { super(s); } }

class TwoException extends Exception {
  public TwoException(String s) { super(s); } }

public class RethrowNew {
  public static void f() throws OneException {
    System.out.println("originating exception in f()");
    throw new OneException("thrown from f()");
  }
  public static void main(String[] args) throws TwoException {
    try { f(); }
      catch(OneException e) {
        System.err.println("Caught in main,e.printStackTrace()");
        e.printStackTrace();
        throw new TwoException("from main");
      }
 }
}
```

```
magnaura.cs.ucy.ac.cy>java RethrowNew
originating the exception in f()
Caught in main, e.printStackTrace()
OneException: thrown from f()
  at RethrowNew.f(RethrowNew.java:13)
  at RethrowNew.main(RethrowNew.java:18)
Exception in thread "main" TwoException: from main
    at RethrowNew.main(RethrowNew.java:23)
```

Αλισυδωτές εξαιρέσεις (exception chaining)

ΕΠΛ233

- 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( ).

Βασικές εξαιρέσεις στη JAVA

ΕΠΛ233

- Η κλάση Throwable περιγράφει ο,τιδήποτε 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.

Runtime Exceptions

ΕΠΛ233

- 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.

Περιορισμοί εξαιρέσεων

ΕΠΛ233

- Όταν υπερσκελίζουμε 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 ό,τι 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

ΕΠΛ233

```
class BaseballException extends Exception {}
class Foul extends BaseballException {}
class Strike extends BaseballException {}
abstract class Inning {
  public Inning() throws BaseballException {}
  public void event() throws BaseballException {}
  public abstract void atBat() throws Strike, Foul;
  public void walk() {}
}
class StormException extends Exception {}
class RainedOut extends StormException {}
class PopFoul extends Foul {}
interface Storm {
  public void event() throws RainedOut;
  public void rainHard() throws RainedOut;
}
```

Περιορισμοί εξαιρέσεων

ΕΠΛ233

```
public class StormyInning extends Inning implements Storm {
// 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:
  //! void walk() throws PopFoul {}
  //Compile error: Interface CANNOT add exceptions to existing
  // methods from the base class:
  //! public void event() throws RainedOut {}
  // 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() {}
```

Περιορισμοί εξαιρέσεων

ΕΠΛ233

```
// Overridden methods can throw inherited exceptions:
public void atBat() throws PopFoul {}

public static void main(String[] args) {
  try {
    StormyInning si = new StormyInning();
    si.atBat();
    }
  catch(PopFoul e) {System.err.println( "Pop foul" );}
  catch(RainedOut e){System.err.println( "Rained out");}
  catch(BaseballException e) {
    System.err.println("Generic" );
  }
```

Περιορισμοί εξαιρέσεων

ΕΠΛ233

```
// Strike not thrown in derived version.
try {// What happens if you upcast?
    Inning i = new StormyInning();
    i.atBat();
// You must catch the exceptions from the
// base-class version of the method:
}
catch (Strike e) { System.err.println( "Strike" ); } catch
(Foul e) { System.err.println( "Foul" ); }
catch (RainedOut e){System.err.println( "Rained out" ); }
catch (BaseballException e) {
    System.err.println( "Generic baseball
exception" );}  }}
```

Παρατηρήσεις

ΕΠΛ233

- It's useful to realize that although exception specifications are enforced by the compiler during inheritance, the exception specifications are not part of the type of a method, which comprises only the method name and argument types. Therefore, you cannot overload methods based on exception specifications.
- Just because an exception specification exists in a base-class version of a method doesn't mean that it must exist in the derived-class version of the method.
  - This is quite different from inheritance rules, where a method in the base class must also exist in the derived class.
- Put another way, the "exception specification interface" for a particular method may narrow during inheritance and overriding, but it may not widen—this is precisely the opposite of the rule for the class interface during inheritance.