Διάλεξη
Εισαγωγή στη Java, Μέρος Β

- Collections & Generics
- Διαχείριση σφαλμάτων στην Java
- Εξωτερικές διεργασίες

- Generics & Collections

Γενικεύσεις – Generics (1)

- Μηχανισμός για υλοποίηση
  - **γενικών κλάσεων**
  - **μεθόδων**
  - **interfaces**
- Χαρακτηριστικό που εμφανίστηκε στην Java SE 5.

https://docs.oracle.com/javase/tutorial/java/generics/why.html

Γενικεύσεις – Generics (2)

Απλή μη γενική κλάση

```
public class Box {                  // any object except primitives e.g. string
        private Object object;
        public void set(Object object) { this.object = object; }
        public Object get() { return object; }
}
```

Γενική κλάση με τύπο αντικειμένου T (T είναι type parameter – class)

```
public class Box<T> {    // T stands for "Type" κλάση ή interface
        private T t;
        public void set(T t) { this.t = t; }
        public T get() { return t; }
}

Box<Integer> integerBox;

Box<Integer> integerBox = new Box<Integer>();
```

https://docs.oracle.com/javase/tutorial/java/generics/index.html

Γενικεύσεις - Generics (3)

Γενικές κλάσης με πολλαπλά αντικείμενα διαφορετικών τύπων

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

Παράδειγμα

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
           this.key = key;
           this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }
}

Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
Pair<String, String>  p2 = new OrderedPair<String, String>("hello", "world");}
OrderedPair<String, Box<Integer> > p = new OrderedPair<>("primes", new Box<Integer>(...));
```

https://docs.oracle.com/javase/tutorial/java/generics/types.html

Γιατί generics;

- Οι γενικεύσεις υλοποιούνται κατά το compile time. Ο compiler
  - πραγματοποιεί τις αντικαταστάσεις στους τύπους
  - εξασφαλίζει τη συμβατότητα τύπων
    - ο προγραμματιστής δεν ανησυχεί για το casting των τύπων
    - μας παρέχει “type-safety at compile time” και κατά συνέπεια να αποφεύγουμε τον κίνδυνο για “ClassCastException” κατά την εκτέλεση
- Επαναχρησιμοποίηση κώδικα π.χ. βιβλιοθήκες
- Βρίσκει εφαρμογή στο Java Collection Framework

Collections (1)

- Πίνακες (Arrays): στατική δομή δεδομένων για αποθήκευση σειράς αντικειμένων
  - Δεν αλλάζει μέγεθος
- Περιπτώσεις: Δεν είναι γνωστό το μέγεθος της δομής δεδομένων
  - Μεγάλο μέγεθος array: σπατάλη μνήμης, χαμηλή απόδοση (loop)
- Ως Collections ορίζονται αντικείμενα **δυναμικών δομών δεδομένων** που ομαδοποιούν άλλα αντικείμενα σε μια οντότητα
- **Java Collection framework (JCF)** = Πλαίσιο & βιβλιοθήκη για τη διαχείριση και αναπαράσταση των collections.
  - Έτοιμες υλοποιήσεις γνωστών δομών δεδομένων και αποδοτικών αλγορίθμων διαχείρισής τους
  - Δομές: Λίστα List, Σύνολο Set, Ουρά Queue, Απεικόνιση Map

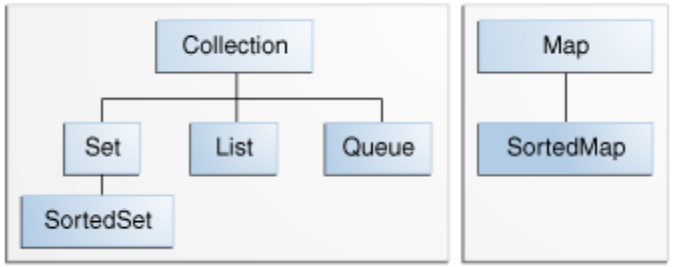
http://docs.oracle.com/javase/tutorial/collections/index.html

Collections (2)

- Ένα collection framework περιλαμβάνει:
  - **Interfaces:** διεπαφή για να χειριστούμε/προσπελαύνουμε με τον ίδιο τρόπο collections, ανεξάρτητα από τις διαφορετικές λεπτομέρειες υλοποίησης τους.
  - **Implementations:** περιλαμβάνουν υλοποιημένες **κλάσεις** των Interfaces. Συνιστούν βιβλιοθήκη για τις εφαρμογές μας.
  - **Αλγορίθμους**: αλγόριθμοι, αποτελούν **μεθόδους**, όπως η αναζήτηση, ταξινόμηση αντικειμένων υλοποιούνται με διαφορετικό τρόπο ως μέθοδοι των διαφόρων κλάσεων. Οι αλγόριθμοι αυτοί είναι **πολυμορφικοί μιας** και η ίδια μέθοδος διαφέρει ανάλογα με το collection interface.
- Μερικά από τα χαρακτηριστικά και προτερήματα από την χρήση collection framework:
  - Έτοιμος κώδικας για χρήση από τις εφαρμογές μας
  - Data structures εμπλουτισμένα με αλγόριθμους για γρήγορη προσπέλαση
  - Ενιαίο API για χρήση, ανεξάρτητο από τις λεπτομέρειες υλοποίησης

Collection interfaces (1)

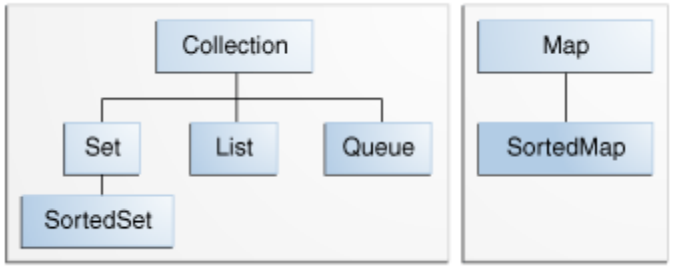
Διαφορετικά είδη **collection interfaces** συνιστούν μια **ιεραρχία** και καθορίζουν διαφορετικούς τύπους αποθήκευσης και χειρισμού αντικειμένων.



- Collection: είναι η ρίζα στην ιεραρχία των collections και δεν υλοποιείται από κάποια κλάση. Χρησιμοποιείται συνήθως για την επικοινωνία collections με γενικό τρόπο.
- Set: ορίζει μια συλλογή, ένα σύνολο, αντικείμενα χωρίς σειρά, στην οποία **δεν μπορούν να υπάρχουν διπλότυπα αντικείμενα**.
- SortedSet: διαθέτει τις ίδιες ιδιότητες με το Set, μόνο που χειρίζεται τα αντικείμενα **με αύξουσα σειρά**.
- List: συνιστά μια συλλογή από αντικείμενα που **εμφανίζουν μια σειρά**. Μπορεί να περιέχει **διπλότυπα**. Μπορούμε να προσπελάσουμε ένα αντικείμενο με χρήση κάποιου index.
- Queue: collection που εμφανίζει τις ιδιότητες της ουράς.

Collection interfaces (2)

- Maps, δεν ανήκουν στη κλάση Collection, περιλαμβάνονται στο JCF



  - Map: collection που αντιστοιχεί κλειδιά με αντικείμενα. Δεν μπορεί να περιέχει διπλότυπα κλειδιά.
  - SortedMap: διαθέτει τις ίδιες ιδιότητες με το Map, καθώς επίσης τα κλειδιά είναι ταξινομημένα.

Collection implementations (1)

Η υλοποίηση των Collection interfaces μπορεί να γίνει με διαφορετικούς τρόπους. Οι υπάρχουσες κατηγορίες συνοψίζονται παρακάτω:

- **General-purpose implementations** σε αυτή την κατηγορία βρίσκουμε τις πιο συχνά χρησιμοποιούμενες κλάσεις για collections.
  - Set → **HashSet**, TreeSet, LinkedHashSet
  - List → **ArrayList**, LinkedList
  - Map → **HashMap**, TreeMap, LinkedHashMap

- Κάθε μια από τις κλάσεις παρέχει υλοποίηση όλων των μεθόδων που έχουν συμφωνηθεί στο interface.
- **Δεν** είναι thread-safe.
- Παρέχουν iterators για προσπέλαση στοιχείων.
- Είναι Serializable και υποστηρίζουν την clone().

- **Wrapper implementations** χρησιμοποιούνται σε συνδυασμό με general-purpose υλοποιήσεις για να παρέχουν επιπλέον λειτουργικότητα.

- **Special-purpose implementations** κλάσεις για ειδικό σκοπό εμφανίζονται στην κατηγορία αυτή που παρουσιάζουν πολλές φορές μη αποδοτική συμπεριφορά.

https://docs.oracle.com/javase/tutorial/collections/interfaces/set.html
https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html
https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html
https://docs.oracle.com/javase/tutorial/collections/implementations/wrapper.html

Collection implementations (2)

- **Concurrent implementations** η υλοποίηση που παρέχεται εξασφαλίζει την ταυτόχρονη προσπέλαση των αντικειμένων αυτής της κατηγορίας από πολυνηματικά περιβάλλοντα (thread safe) (java.util.concurrent package).
- **Convenience implementations** collections που είναι σχεδιασμένα με static συμπεριφορά (πχ singleton sets).
- **Abstract implementations** παρέχουν abstract κλάσεις για custom υλοποίηση από τον χρήστη.

Προσπέλαση ενός Collection (1)

- Υπάρχουν δυο τρόποι για να προσπελάσει κανείς κάποιο collection αντικείμενο, και κατά συνέπεια τα στοιχεία που το αποτελούν, ανεξάρτητα από τις λεπτομέρειες υλοποίησης:
  - Με χρήση της δομής ελέγχου for-each
  - Με χρήση Iterator
- Η δομή ελέγχου **for-each** μας επιτρέπει να προσπελάσουμε με συνέπεια το σύνολο των στοιχείων ενός collection (ή ενός πίνακα array).
  - for (Object o : collection) System.out.println(o);

https://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html

Προσπέλαση ενός Collection (2)

- Ο Iterator είναι ένα interface που ορίζει τις μεθόδους προς υλοποίηση για να προσπελάσουμε ένα collection καθώς και να διαγράψουμε στοιχεία του
  - Κάθε Collection υλοποιεί τη μέθοδο **iterator()** μας επιστρέφει ένα αντικείμενο τύπου Iterator

```
public interface Iterator<E> {
      boolean hasNext();
      E next();
      void remove(); //optional
      }
```

- Η μέθοδος **hasNext** επιστρέφει true εάν υπάρχουν και άλλα στοιχεία στο collection
- Η μέθοδος **next** επιστρέφει το επόμενο στοιχείο.
- Η **remove** διαγράφει το τελευταίο στοιχείο που έχει επιστρέψει η next.

```
static void filter(Collection<?> c) {
      for (Iterator<?> it = c.iterator(); it.hasNext(); )
                  if (!cond(it.next()))
                              it.remove();
}
```

wildcard character ('?') represents an unknown type

https://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html
https://docs.oracle.com/javase/tutorial/java/generics/wildcards.html

Άλλα χρήσιμα χαρακτηριστικά...

- Bulk operations
  - containsAll(): ελέγχουμε αν ένα collection περιέχει όλα τα στοιχεία ενός άλλου.
  - addAll(): «συνενώνει» δυο collections.
  - removeAll(): αφαιρεί τα στοιχεία από ένα collection που εμφανίζονται σε ένα άλλο (διαφορά συνόλων).
  - retainAll(): η αντίστροφη διαδικασία από το removeAll, αφαιρεί τα στοιχεία από ένα collection που ΔΕΝ εμφανίζονται σε ένα άλλο (διαφορά συνόλων).
  - Clear(): αδειάζουμε ένα collection από τα στοιχεία του.
- size(): Μας επιστρέφει το πλήθος των στοιχείων που διατηρεί το collection
- toArray()
  - Μέθοδος που μας επιτρέπει να μετατρέψουμε τα στοιχεία ενός collection σε πίνακα από αντικείμενα

```
Collection<String> c

Object[] a = c.toArray();

ή και

String[] a = (String[]) c.toArray(new String[c.size()]);
```

https://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html

Generics @ Collections

- Τα generics είναι άμεσα συνδεδεμένα με την χρήση Collections

```
List<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

αντί

```
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
```

http://docs.oracle.com/javase/tutorial/java/generics/index.html

Η χρήση των Generics μέσα από παραδείγματα

```
Set setOfRawType = new HashSet<String>();
Set setOfRawType = new HashSet<Integer>();
```

```
Set<Object> setOfAnyType = new HashSet<Object>();
setOfAnyType.add("abc");
setOfAnyType.add(new Float(3.0f));
```

```
Set<?> setOfUnknownType = new LinkedHashSet<String>();
setOfUnknownType = new LinkedHashSet<Integer>();
```

```
Set<String> setOfString = new HashSet<String>();

setOfString = new LinkedHashSet<String>();
```

```
Set<Object> SetOfObject = new HashSet<String>(); //compiler error
```

https://github.com/Shekharrajak/Java-Generic-and-Collections/wiki/Generic-in-java

- Διαχείριση σφαλμάτων στην Java

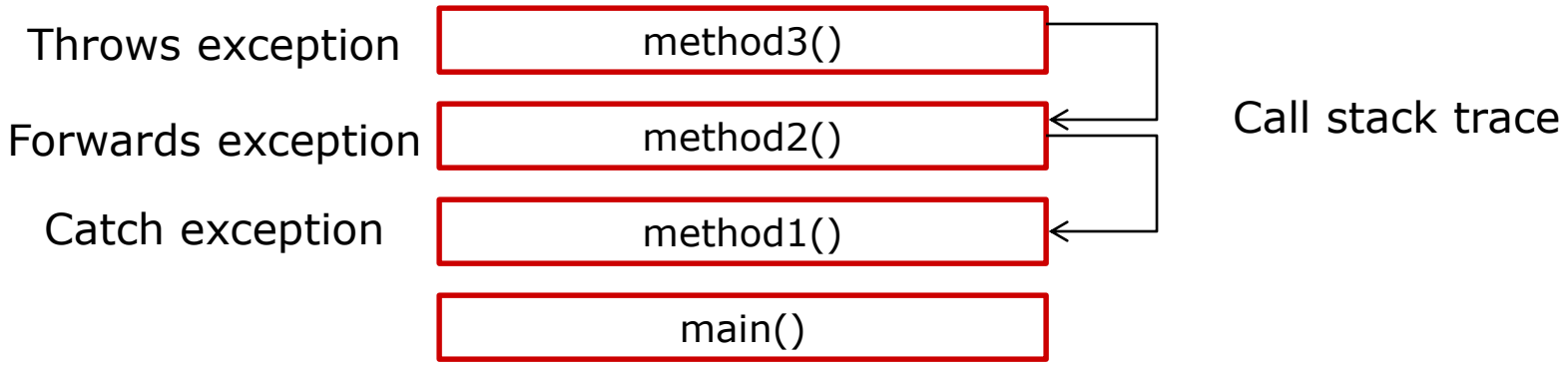
Σφάλματα στην Java (1/2)

- Τα σφάλματα (Bugs) είναι το πλέον κοινότυπο πρόβλημα ενός προγράμματος
  - Προγραμματιστικά λάθη
  - Λάθη σχεδιασμού
  - Λάθος είσοδος δεδομένων
  - Αστοχία υλικού
  - Μη ελεγχόμενοι, εξωγενείς παράγοντες
- Κάθε προγραμματιστής πρέπει να διαχειρίζεται τις καταστάσεις:
  - Στην C: Κάθε συνάρτηση γυρνά κάποια τιμή ενώ παράλληλα η perror τυπώνει το σφάλμα που συνέβη.
  - Στην Java: Χρήση εξαιρέσεων (exceptions). Υπάρχουν δύο είδη:
    1. οι **ελεγχόμενες εξαιρέσεις (checked)** και
    2. **μη ελεγχόμενες (unchecked)**

https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

Σφάλματα στην Java (2/2)

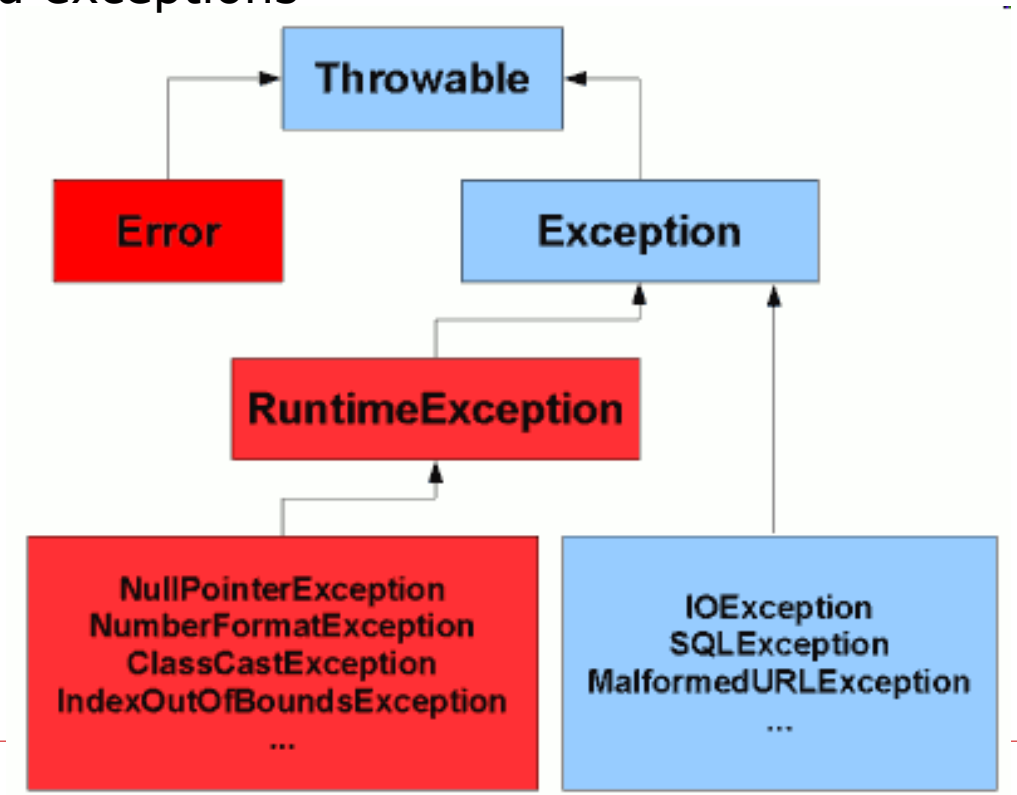
- Κατά την εκτέλεση ενός προγράμματος όταν συμβεί σφάλμα σε κάποια μέθοδο που εκτελείται, **αυτή δημιουργεί ένα αντικείμενο** (exception object) και το παραδίδει στο runtime system (κάνει **throw ένα exception**).
- Ένα **exception αντικείμενο** περιέχει πληροφορίες για
  - το είδος του σφάλματος που συνέβει,
  - την κατάσταση του προγράμματος μας τη στιγμή του exception κ.α...
- Το runtime system *ψάχνει* για κάποιο **handler** ορισμένο από τον προγραμματιστή για να διαχειρίζεται αυτό το σφάλμα (κάνει **catch το exception**).
- Η αναζήτηση ξεκινά ιεραρχικά από την τρέχουσα συνάρτηση που συνέβει το σφάλμα και φτάνει μέχρι την main.



Κατηγορίες exceptions

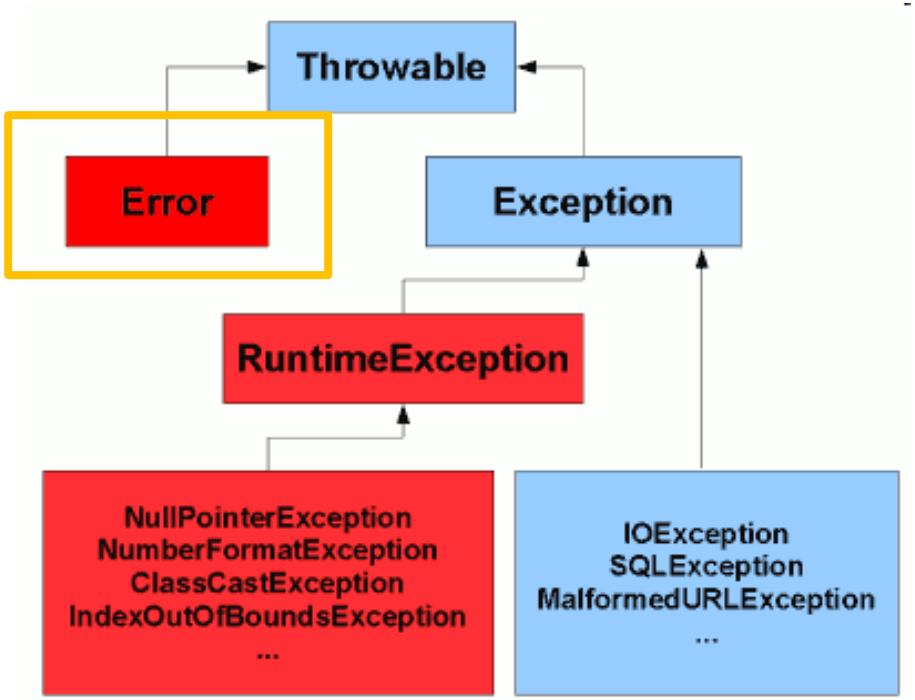
Η Java μας παρέχει **τριών ειδών exceptions.** Όλα έχουν ως υπερκλάση την java.lang.Throwable

1. Error
2. Runtime exceptions/Unchecked exceptions
3. Checked exceptions



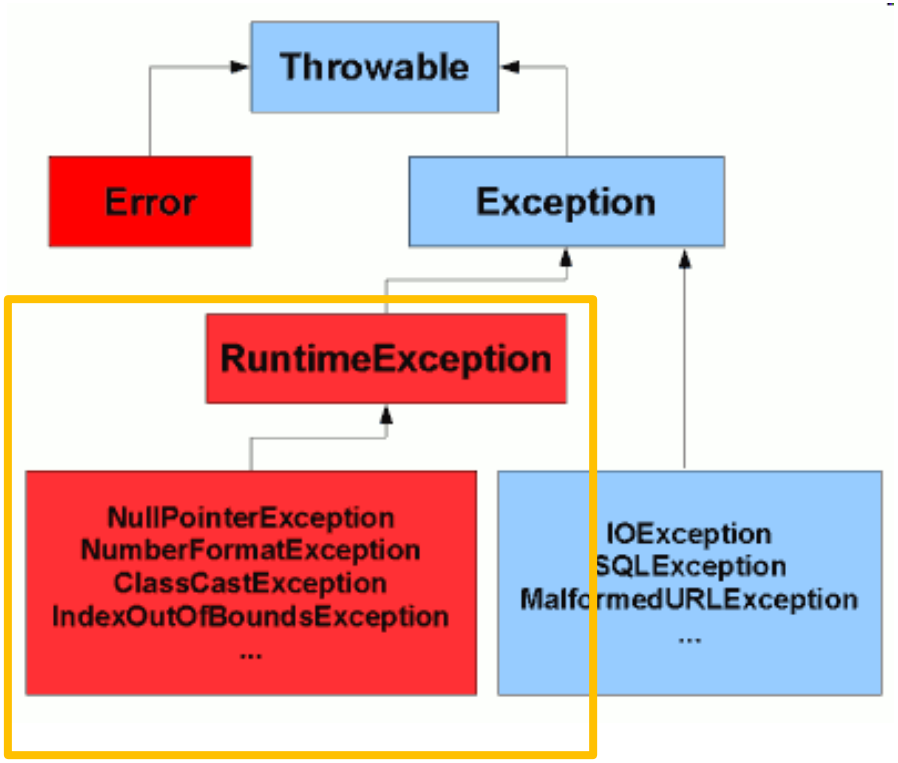
Error exceptions

- Ορίζουν exceptional conditions που οφείλονται σε **εξωτερικούς παράγοντες** από την εφαρμογή
  - δεν μπορεί να προβλέψει
  - να κάνει recover
  - π.χ. αστοχία στο hardware
- Το stack trace μέχρι το σημείο που συνέβει το exception
- Το πρόγραμμα μας τερματίζει
- Δεν υπόκεινται στο “Catch or Specify Requirement”
- Δεν έχει νόημα να γραφεί κώδικας για να το χειριστεί το πρόγραμμα ένα Error exception



Runtime exceptions / Unchecked exceptions

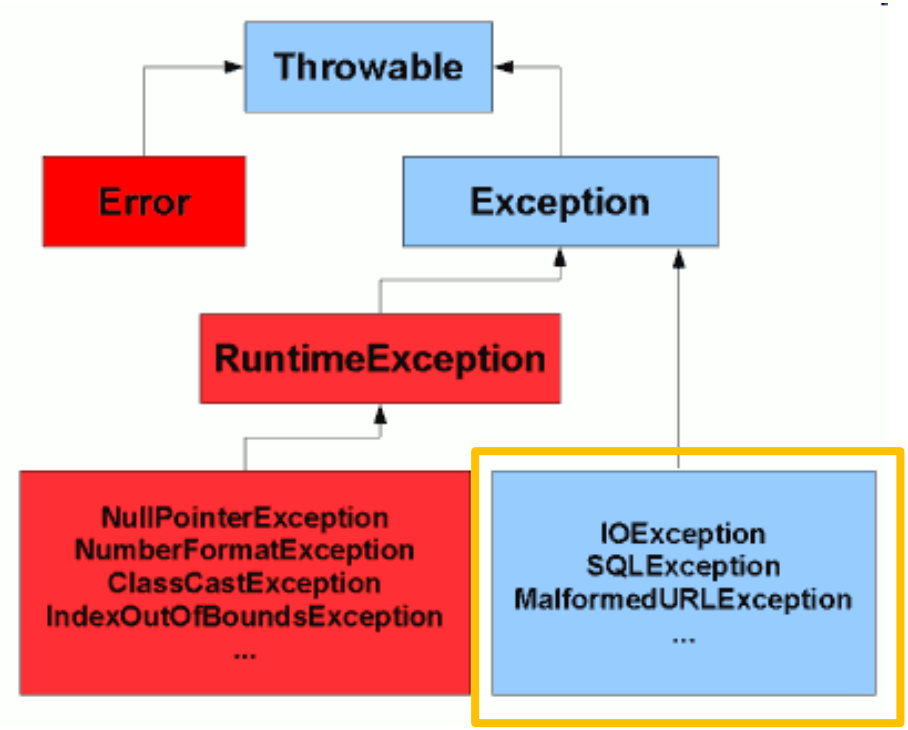
- Ορίζουν exceptional conditions που οφείλονται σε λογικά **λάθη του προγραμματιστή**
- Το πιο δημοφιλές exception σε αυτή την κατηγορία είναι το **NullPointerException**!!
- Δεν απαιτείται να υπόκεινται στο "Catch or Specify Requirement" και ανήκουν στην κλάση RunTimeException
  - Μπορεί να γραφεί κώδικας για να το χειριστεί (θα απαιτηθεί σε πολλά σημεία και ο κώδικας δεν θα είναι πλέον ευκρινής)
  - **Επιστρέφεις και διορθώνεις το κώδικα**



https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

Checked exceptions

- Ορίζουν exceptional conditions όπου μια καλογραμμένη εφαρμογή θα πρέπει να
  - προβλέψει
  - κάνει recover
- Υπόκεινται στο "Catch or Specify Requirement"
  - το πρόγραμμα μας θα πρέπει να διαθέτει handlers για να τα κάνει catch και να δώσει προς τον χρήστη κάποιο διαγνωστικό μήνυμα
- Όλα τα exceptions θεωρούνται checked, εκτός από που ανήκουν στις κλάσεις Error, RunTimeException και τις αντίστοιχες υποκλάσεις τους



Exception handler

Εάν δεν θέλουμε να διαχειριστούμε κάποιο/α "ExceptionType", τότε εισάγουμε στο header της εκάστοτε μεθόδου την δήλωση … **throws** ExceptionType

Η Java μας παρέχει τρία exception handler components. Τα **try**, **catch**, και **finally** blocks μας επιτρέπουν να ορίσουμε τον δικό μας exception handler.

```
try
{
      code
}
catch (ExceptionType name) {
       System.err.println(.....);
      ....
}
catch (ExceptionType name) {
       System.err.println(.....);
      ....
} finally {
      .....
}
```

Περικλύω μέσα στο try block κομμάτι της εφαρμογής μου που υπάρχει πιθανότητα να γίνει throw ένα exception.

Κάθε catch block ορίζει έναν exception handler, για ένα δεδομένο τύπο exception. Συνήθως τυπώνουν διαγνωστικό μήνυμα προς το χρήστη.

Το finally block εκτελείται πάντα, είτε συμβεί exception είτε όχι. Συνήθως γράφουμε σ' αυτό clean up κώδικα.

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Vector;

public class ExceptionsExample {

    public void readFile(String path){...}

    public void writeFile(String path){...}

    public void ArrayIndexOutOfBoundExceptionExample(){...}

    public void ClassCastExceptionExample() throws ClassCastException{...}

    public static void main(String args[]){...}

}
```

```
public void writeFile(String path){
    try {
        RandomAccessFile f = new RandomAccessFile(path, "rw");
        String s = "foo";
        f.writeBytes(s);
        System.out.println("Wrote string:"+s);
        f.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Η ανώτερη ιεραρχικά κλάση διαχειρίζεται και τα 2 πιθανά σφάλματα.

Το σφάλμα θα το διαχειριστεί η καλούσα μέθοδος.

```
public void readFile(String path){
    try {
        RandomAccessFile f = new RandomAccessFile(path, "r");
        String s = f.readLine();
        System.out.println("Read string:"+s);
        f.close();
    }catch (FileNotFoundException ex) {
        ex.printStackTrace();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Χρήση 2 διαδοχικών exceptions για να καλύψουμε τις περιπτώσεις απώλειας του αρχείου και σφάλματος εγγραφής.

Γιατί θα έχουμε πρόβλημα εδώ;

```
public void ArrayIndexOutOfBoundExceptionExample(){
    int[] a = new int[10];
    for(int i=0;i<=10;i++)
        a[i] = i+1;
}

public void ClassCastExceptionExample() throws ClassCastException{
    Vector v = new Vector();
    v.add(new Integer(1));
    double b = ((Double)v.elementAt(0)).doubleValue();
}
```

```
47  public static void main(String args[]){
48      ExceptionsExample ee = new ExceptionsExample();
49      ee.writeFile("test.txt");
50      ee.readFile("test.txt");
51
52      try{
53          ee.ClassCastExceptionExample();
54      }catch(ClassCastException e){
55          e.printStackTrace();
56      }
57
58      ee.ArrayIndexOutOfBoundExceptionExample();
59
60
61  }
```

Η χρήση του stack trace που μας δίνει κάθε exception επιτρέπει την αποσφαλμάτωση του προγράμματός μας. Η διαδικασία είναι η εξής:

1. Αναζητούμε την **πρώτη** δικιά μας κλήση που δημιούργησε πρόβλημα (από πάνω προς τα κάτω).
2. Ανατρέχουμε στην αντίστοιχη γραμμή του κώδικά μας. Το σημείο αυτό έχει πρόβλημα.
3. Ανάλογα με το πρόβλημα ελέγχουμε την είσοδο ή την έξοδο της εντολής.

Διαχειριζόμαστε το exception όπως το ζητά η αντίστοιχη μέθοδος.

Σφάλμα στην γραμμή 53! Μπορούσε να μην εμφανιστεί αν δεν ζητούσαμε εκτύπωση.

```
Wrote string: foo
Read string: foo
java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.Double
        at course2.ExceptionsExample.ClassCastExceptionExample(ExceptionsExample.java:44)
        at course2.ExceptionsExample.main(ExceptionsExample.java:53)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at course2.ExceptionsExample.ArrayIndexOutOfBoundExceptionExample(ExceptionsExample.java:38)
        at course2.ExceptionsExample.main(ExceptionsExample.java:58)
Java Result: 1
```

Σφάλμα στην γραμμή 58! Προέρχεται από λάθος κλήση στην γραμμή 38.

```
a[i] = i+1;
```

Αποσφαλμάτωση με χρήση Exceptions – Πλεονεκτήματα (1/2)

- Η κλάση exceptions προσφέρει την δυνατότητα εκτύπωσης των κλήσεων που οδήγησαν στο συγκεκριμένο σφάλμα
  - Μέθοδος **printStackTrace();**
  - Τυπώνονται οι διαδοχικές κλήσεις μεθόδων που οδήγησαν στο συγκεκριμένο σφάλμα.
- ***ΔΕΝ ΕΙΝΑΙ ΚΑΛΗ ΠΡΑΚΤΙΚΗ:***
  - Να χρησιμοποιείται η ανώτερη ιεραρχικά κλάση για την διαχείριση όλων των σφαλμάτων (δηλαδή η Exception).
  - Να τοποθετείται throws στην μέθοδο main.

Αποσφαλμάτωση με χρήση Exceptions – Πλεονεκτήματα (2/2)

- Μερικά από τα προτερήματα που μας παρέχει η χρήση exceptions:
  - Διαχωρισμός του κώδικα που κάνει exception handling από τον κώδικα της εφαρμογής μας.
  - Διαχείριση ενός exception υψηλότερα στο call stack, απλά με την χρήση της δήλωσης throws.
  - Κατηγοριοποίηση και χειρισμός των exceptions σε ομάδες λόγω της ύπαρξης ιεραρχίας κλάσεων
    - πχ τα exceptions FileNotFoundException και InterruptedIOException μπορούν να αντικατασταθούν από το IOException

- Εξωτερικές διεργασίες

Εξωτερικές διεργασίες (1)

- Η Java σχεδιάστηκε με σκοπό να παρέχει στις εφαρμογές ανεξαρτησία από το περιβάλλον εκτέλεσης.
- Όμως, υπάρχουν περιπτώσεις που θέλουμε να εκτελέσουμε μια εφαρμογή που να σχετίζεται με το λειτουργικό σύστημα που βρισκόμαστε.
- Η μεθοδολογία που ακολουθούμε είναι να δημιουργούμε μια νέα διεργασία μέσα στην εφαρμοφή μας που θα εκτελεί platform dependent ενέργειες.

Εξωτερικές διεργασίες (2)

- Η **κλάση Runtime** αποτελεί μια διεπαφή με το περιβάλλον που τρέχει η εφαρμογή μας
  - **exec(String[] cmdAndArgs):** *δημιουργεί νέα διεργασία* που εκτελεί την εντολή με παραμέτρους που της περνάμε σαν όρισμα
  - **getRuntime():** μας *επιστρέφει ένα αντικείμενο* τύπου **Runtime** για την ίδια μας την εφαρμογή

    Process p = Runtime.getRuntime().exec("/bin/ls");
- Προσοχή!! Η java ανακατευθύνει τα stdin, stdout, stderr.
  - Για να αλληλεπιδράσουμε με την διεργασία πρέπει να χρησιμοποιήσουμε τις μεθόδους
    - getInputStream()
    - getOutputStream()
    - getErrorStream()
  - που μας παρέχει το αντικείμενο τύπου Process.

Παράδειγμα

```
class ExecInput {
       public static void main(String argv[]) {
        try {
            String line;

            Process p = Runtime.getRuntime().exec("/bin/ls -aFl");


           BufferedReader input = new BufferedReader(new InputStreamReader(p
                   getInputStream()));

            while ((line = input.readLine()) != null) {
                System.out.println(line);
            }
           input.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Εκτελούμε την ls που βρίσκεται στον κατάλογο /bin

Η getInputStream μας παρέχει πρόσβαση στο output stream της διεργασίας παιδιού. Τα δεδομένα αυτά αποτελούν input stream για την main

Κάνουμε buffered το input stream και διαβάζουμε γραμμή-γραμμή τα δεδομένα

Κλείνουμε πάντα τις ροές!!