Exceptions in Java

Εξαιρέσεις στην Java

Τι είναι μια εξαίρεση

- Έστω το τμήμα κώδικα:

```
int age = Integer.parseInt(input);
```

  Τι θα συμβεί αν ο χρήστης εισάγει μια μη έγκυρη τιμή ηλικίας;
- Έχουμε γράψει κώδικα για να ανοίξουμε ένα αρχείο και το αρχείο αυτό δεν μπορεί να εντοπιστεί
- Μια σύνδεση δικτύου χάνεται ξαφνικά ή η JVM δεν έχει άλλη διαθέσιμη μνήμη.

- Αν δεν χειριστούμε αυτές τις ΕΞΑΙΡΕΣΕΙΣ η εκτέλεση του προγράμματος διακόπτεται με μη φυσιολογικό τρόπο
- ARRIANE 5 failure?

https://www.slideshare.net/software-engineering-book/ariane5failure-pres

Τι είναι μια εξαίρεση

- Ορισμένα πράγματα μπορεί να εξελιχθούν με λάθος τρόπο κατά τη διάρκεια της εκτέλεσης και δεν μπορούν να ανιχνευθούν κατά τη διάρκεια της μεταγλώττισης
- Ένα άλλο παράδειγμα: απόπειρα διαίρεσης με το `0`
  - Από την πλευρά του compiler δεν υπάρχει κανένα πρόβλημα με τις αντίστοιχες εντολές και τα προβλήματα θα προκύψουν μόνο κατά την εκτέλεση του προγράμματος
  - Στο σημείο αυτό «ενεργοποιείται ένας συναγερμός» και η Java προσπαθεί να «παράγει μια εξαίρεση» υποδηλώνοντας ότι κάτι αντικανονικό έχει συμβεί.

```
import java.util.*;

public class StackDemo {
   public static void main(String args[]) {
   // creating stack
   Stack st = new Stack();

   // populating stack
   st.push("Java");
   st.push("Source");

   // removing top object
   System.out.println("Removed object is: "+st.pop());
    //System.out.println("Removed object is: "+st.pop());
   //System.out.println("Removed object is: "+st.pop());
   // elements after remove
   System.out.println("Elements after remove: "+st);
   }
}
```

```
Removed object is: Source
Elements after remove: [Java]
```

*The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). Java collection framework provides many interfaces and classes to store the collection of objects. One of them is the **Stack class** that provides different operations such as push, pop, search, etc. (https://www.javatpoint.com/java-stack)*

```
import java.util.*;
public class StackDemo {
   public static void main(String args[]) {
   // creating stack
   Stack st = new Stack();

   // populating stack
   st.push("Java");
   st.push("Source");

   // removing top object
   System.out.println("Removed object is: "+st.pop());
    System.out.println("Removed object is: "+st.pop());
   //System.out.println("Removed object is: "+st.pop());
   // elements after remove
   System.out.println("Elements after remove: "+st);
   }
}
```

```
Removed object is: Source
Removed object is: Java
Elements after remove: []
```

```
import java.util.*;
public class StackDemo {
   public static void main(String args[]) {
   // creating stack
   Stack st = new Stack();
   // populating stack
   st.push("Java");
   st.push("Source");
// removing top object
   System.out.println("Removed object is: "+st.pop());
   System.out.println("Removed object is: "+st.pop());
   System.out.println("Removed object is: "+st.pop());
   // elements after remove
   System.out.println("Elements after remove: "+st);
   }
}
```

Παράδειγμα 1

```
public static void main(String[ ] args) {

        Stack stack = new Stack();

        stack.push("Nick");
        stack.push("Bob");

        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println("This statement will not be printed");

}
```

*The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). Java collection framework provides many interfaces and classes to store the collection of objects. One of them is the **Stack class** that provides different operations such as push, pop, search, etc. (https://www.javatpoint.com/java-stack)*

Παράδειγμα 2

```
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;


    result1 = num1/(num2-num3);
    System.out.println("Result1 = " +result1);

  }}
```

Ορολογία: Actors and Actions

- **Operation - Λειτουργία**
  Μια **μέθοδος** που μπορεί να παράγει (raise) μια εξαίρεση.
- **Invoker**
  Μια **μέθοδος** που καλεί λειτουργίες και χειρίζεται τις εξαιρέσεις που προκύπτουν.
- **Exception**
  Μια ακριβής και πλήρης περιγραφή ενός αντικανονικού γεγονότος. Πρόκειται για αντικείμενο στη Java.

- **Raise (Παραγωγή Εξαίρεσης)**
  Αποστολή μιας εξαίρεσης από την operation στον invoker. (Καλείται **throw** στην Java).
- **Handle - Χειρισμός**
  Η απόκριση του Invoker στην εξαίρεση, καλείται **catch** στη Java.

Ορισμένοι τύποι εξαιρέσεων

Arithmetic Exception	πρόβλημα κατά την αποτίμηση αριθμητικής παράστασης, όπως η διαίρεση με το μηδέν
NullPointerException	κλήση μεθόδου μέσω αναφοράς που έχει τιμή null
IndexOutOfBoundsException	ένας δείκτης πίνακα έχει βρεθεί εκτός των ορίων της δομής
EOFException	εντοπίστηκε σύμβολο τέλους αρχείου

Εξαιρέσεις

Ένα πρόγραμμα μπορεί να αντιμετωπίσει μια εξαίρεση με τρεις τρόπους

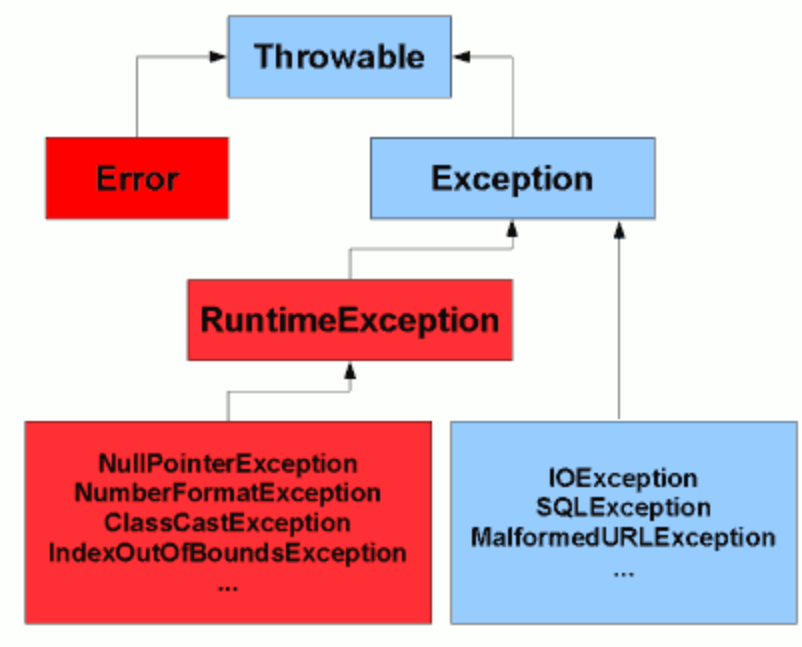
1. να την αγνοήσει
2. να την χειριστεί στο σημείο που παράγεται
3. να την χειριστεί σε κάποιο άλλο σημείο του προγράμματος

When a dynamic linking failure or other hard failure in the Java virtual machine occurs, the virtual machine throws an Error. Simple programs typically do *not* catch or throw Errors.

Κατηγοριοποίηση Εξαιρέσεων Java

- **Unchecked Exceptions (**checked at runtime)
  Δεν είναι υποχρεωτικό να χειριστούμε αυτούς τους τύπους εξαιρέσεων.
  - *Runtime exceptions* can be generated by methods or by the JVM itself.
  - *Errors* are generated from deep within the JVM, and often indicate a truly fatal state.

- **Checked Exceptions (**checked at compile-time)
  Πρέπει είτε να «συλληφθούν» (caught) από μια μέθοδο ή να δηλωθούν στην υπογραφή της



When a dynamic linking failure or other hard failure in the Java virtual machine occurs, the virtual machine throws an Error.
Simple programs typically do not catch or throw Errors.

Keywords for Java Exceptions

- **throws**
  Περιγράφει τις εξαιρέσεις που μπορεί να παραχθούν από μια μέθοδο.
  (στην υπογραφή της μεθόδου)
- **throw**
  Παραγωγή μιας εξαίρεσης και προώθηση στον πρώτο διαθέσιμο χειριστή στην στοίβα κλήσεων.
- **try**
  Υποδηλώνει την αρχή ενός block που σχετίζεται με ένα σύνολο χειριστών εξαιρέσεων (μπορεί να παράγει εξαιρέσεις).
- **catch**
  Αν το block που περιλαμβάνεται σε ένα try παράγει μια εξαίρεση του αντίστοιχου τύπου, η ροή του ελέγχου μεταφέρεται εδώ.
- **finally**
  Καλείται όταν τερματίζεται το τμήμα try και μετά από οποιοδήποτε τυχόν χειρισμό στο τμήμα catch.

Try – catch και finally

TRY: ορίζει το μπλοκ κώδικα που μπορεί να προκαλέσει exception. Το μπλοκ αυτό ονομάζεται guarded region.

CATCH: δηλώνει το είδος εξαίρεσης που χειρίζεται και σε περίπτωση που συμβεί εκτελείται ο κώδικας που περιέχει.

FINALLY: περιέχει κώδικα που εκτελείται πάντα είτε προκλήθηκε εξαίρεση, είτε όχι. Π.χ. κλείσιμο αρχείων, συνδέσεων με ΒΔ, network sockets, κτλ.

- Ένα block catch πρέπει να ακολουθεί ένα try block
- Τα catch blocks προηγούνται του finally block
- Το finally block είναι προαιρετικό
- Ένα try block δε μπορεί να μην ακολουθείτε είτε από ένα catch είτε από ένα finally.
- Δεν βάζουμε κώδικα ανάμεσα στα try, catch και finally blocks.

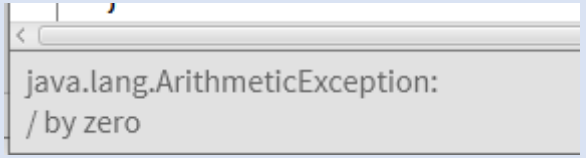
Γενική σύνταξη

```
try { // Protected code or ‘guarded region’ }

catch (ExceptionType1 e1)
  { // Catch block }
catch (ExceptionType2 e2)
  { // Catch block }
catch (ExceptionType3 e3)
  { // Catch block }

finally
  { // The finally block always executes }
```

Παράδειγμα 2 (ξανά)

```
import java.io.IOException;
class TesteExceptions
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;
    result1 = num1/(num2-num3);
    System.out.println("Result1 = " +result1);
      tion has been thrown!");}
  } }
```

Τι θα συμβεί;



Παράδειγμα 2 β

```
import java.io.IOException;
class TesteExceptions
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;
    try
        {
         result1 = num1/(num2-num3);
        System.out.println("Result1 = " +result1);
        }
   catch (ArithmeticException g)
        {System.out.println("Division by zero");}
    catch (Exception e)
        {System.out.println("An Exception has been thrown!");}
  } }
```

Τι θα συμβεί;

Division by zero

Παράδειγμα 2 γ

```
import java.io.IOException;
class TesteExceptions
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;
    try
          {
          result1 = num1/(num2-num3);
          System.out.println("Result1 = " +result1);
          }
   catch (ArithmeticException g)
          {System.out.println("Division by zero");}
    catch (Exception e)
          {System.out.println("An Exception has been thrown!");}
   finally {System.out.println("no matter what you pass from here!");}
  } }
```

Τι θα συμβεί;

Division by zero

Παράδειγμα 2 δ

```
import java.io.IOException;
class TesteExceptions
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;
    try
        {
        result1 = num1/(num2-num3);
        System.out.println("Result1 = " +result1);
        }
   catch (ArithmeticException g)
        {System.out.println("Division by zero");}
    catch (Exception e)
        {System.out.println("An Exception has been thrown!");}
   finally {System.out.println("no matter what you pass from here!");}
  } }
```

Τι θα συμβεί;

Division by zero
no matter what you pass from here!

Αν num3=0

Result1 = 2
no matter what you pass from here!

Παράδειγμα 2ε

TesteExceptions

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
import java.io.IOException;
class TesteExceptions
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;

    try
    {
        result1 = num1/(num2-num3);
        System.out.println("Result1 = " +result1);
    }
    catch (ArithmeticException g)
    {System.out.println("Division by zero");}
    catch (Exception e)
    {System.out.println("An Exception has been thrown!");}


   }
  }
```

Παράδειγμα 2ζ

```
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;

    try
    {
    result1 = num1/(num2-num3);
    System.out.println("Result1 = " +result1);
    }
    catch (Exception e)
    {System.out.println("An Exception has been thrown!");}
    catch (ArithmeticException g)
    {System.out.println("Division by zero");}
    finally {System.out.println("no matter what you pass from here!
  }
  }
```

```
exception java.lang.ArithmeticException has already been
caught
```

Παράδειγμα 3α

```
public void foo() {
  try {
       int a[] = new int[2];
       a[4] = 1;
  } catch (ArrayIndexOutOfBoundsException e) {
       System.out.println("exception: " + e.getMessage());
       e.printStackTrace();
  }
}


public int[] bar() {
  int a[] = new int[2];
  for (int x = 0; x <= 2; x++) { a[x] = 0; }
  return a;}
```

Παράδειγμα 3β

```
public void foo() {
  try { /* marks the start of a try-catch block */
        int a[] = new int[2];
        a[4] = 1; /* causes a runtime exception due to the index
  */
  } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("exception: " + e.getMessage());
        e.printStackTrace();
  }
}
/* This code also compiles, but throws an exception at
  runtime! It is both less obvious and more common */
public int[] bar() {
  int a[] = new int[2];
  for (int x = 0; x <= 2; x++) { a[x] = 0; }
  return a;}
```

Πλεονεκτήματα των Exceptions1

Από "The Java Tutorials"
http://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html

Πλεονέκτημα 1: Διαχωρισμός κώδικα χειρισμού σφαλμάτων από «κανονικό» κώδικα (αποφυγή "spaghetti" code)

Πλεονεκτήματα των Exceptions 2

Έστω το παρακάτω τμήμα κώδικα

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

Ο κώδικας φαίνεται αρκετά «απλός» και καθαρός αλλά αγνοεί τα παρακάτω πιθανά σφάλματα:

- Τι θα συμβεί αν το αρχείο δεν μπορεί να ανοίξει?
- Τι θα συμβεί αν το μήκος του αρχείου δεν μπορεί να προσδιοριστεί?
- Τι θα συμβεί εάν δεν μπορεί να δεσμευθεί η απαραίτητη μνήμη?
- Τι θα συμβεί εάν η ανάγνωση του αρχείου αποτύχει?
- Τι θα συμβεί εάν το αρχείο δεν μπορεί να κλείσει?

Πλεονεκτήματα των Exceptions 3α

Θα μπορούσαμε να γράψουμε κώδικα για τον χειρισμό και την αναφορά όλων των πιθανών προβλημάτων ως εξής:

```
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

Υπάρχει τόσος κώδικας για τον χειρισμό των σφαλμάτων που οι αρχικές 7 γραμμές κώδικα έχουν «χαθεί»

Πλεονεκτήματα των Exceptions 3β

Θα μπορούσαμε να γράψουμε κώδικα για τον χειρισμό και την αναφορά όλων των πιθανών προβλημάτων ως εξής:

```
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
            }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

Υπάρχει τόσος κώδικας για τον χειρισμό των σφαλμάτων που οι αρχικές 7 γραμμές κώδικα έχουν «χαθεί»

Πλεονεκτήματα των Exceptions 4

Ο μηχανισμός των εξαιρέσεων επιτρέπει να διατηρήσουμε ανέπαφη τη λογική του κυρίως κώδικα και να χειριστούμε τα σφάλματα σε άλλο σημείο:

```
readFile {
    try {

        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;

    } catch (fileOpenFailed) {
       doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Πλεονεκτήματα των Exceptions 5

Πλεονέκτημα 2: «Προώθηση» των σφαλμάτων προς τα πάνω στην στοίβα κλήσεων

Έστω ότι η readFile είναι η 4η μέθοδος στη σειρά σε μια ακολουθία κλήσεων μεθόδων από το κυρίως πρόγραμμα (method1->method2->method3->readFile)

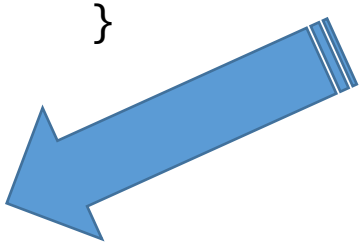
Έστω ότι η method1 είναι η μόνη μέθοδος που ενδιαφέρεται για τα σφάλματα που μπορεί να συμβούν στην readFile

Η συμβατική αντιμετώπιση θα ήταν

Πλεονεκτήματα των Exceptions 6

```
method1 {
    errorCodeType error;
    error = call method2;
    if (error)
        doErrorProcessing;
    else
        proceed;
}
```

```
errorCodeType method2 {
    errorCodeType error;
    error = call method3;
    if (error)
        return error;
    else
        proceed;
}
```



```
errorCodeType method3 {
    errorCodeType error;
    error = call readFile;
    if (error)
        return error;
    else
        proceed;
}
```

Πλεονεκτήματα των Exceptions 7

Αντιμετώπιση με τον μηχανισμό των exceptions

```
method1 {
    try {
        call method2;
    } catch (exception e) {
        doErrorProcessing;
    }
}
```

```
method2 throws exception {
    call method3;
}

method3 throws exception {
    call readFile;
}

readFile throws exception {
    . . . ;
}
```

Πλεονεκτήματα των Exceptions 8

Πλεονέκτημα 3: Κατηγοριοποίηση και Διαφοροποίηση βάσει τύπου των σφαλμάτων

Όλες οι εξαιρέσεις είναι αντικείμενα κλάσεων

Μια μέθοδος μπορεί να γράψει κώδικα για τον χειρισμό μιας συγκεκριμένης κατηγορίας σφαλμάτων

```
catch (FileNotFoundException e)
{
    ...
}
```

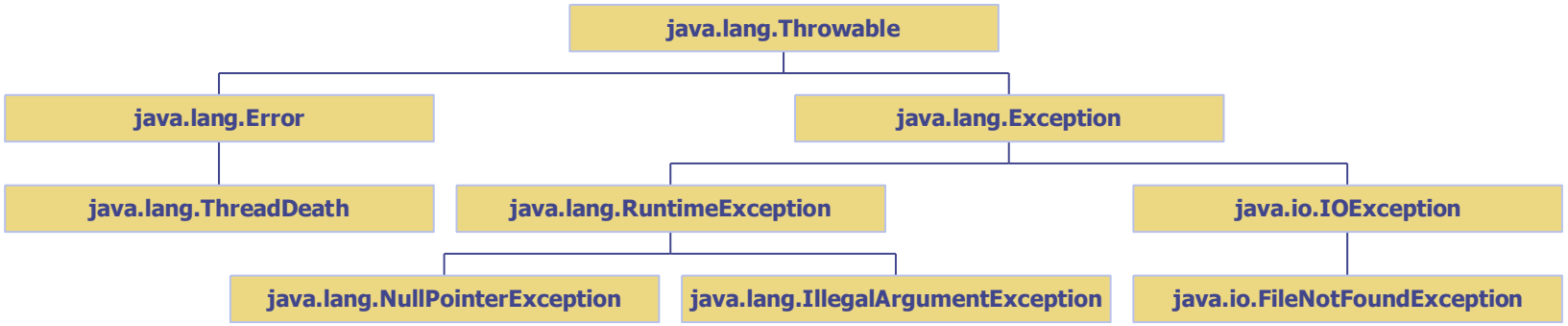
Χειρισμός μόνο σφαλμάτων που σχετίζονται με αδυναμία εύρεσης του αρχείου

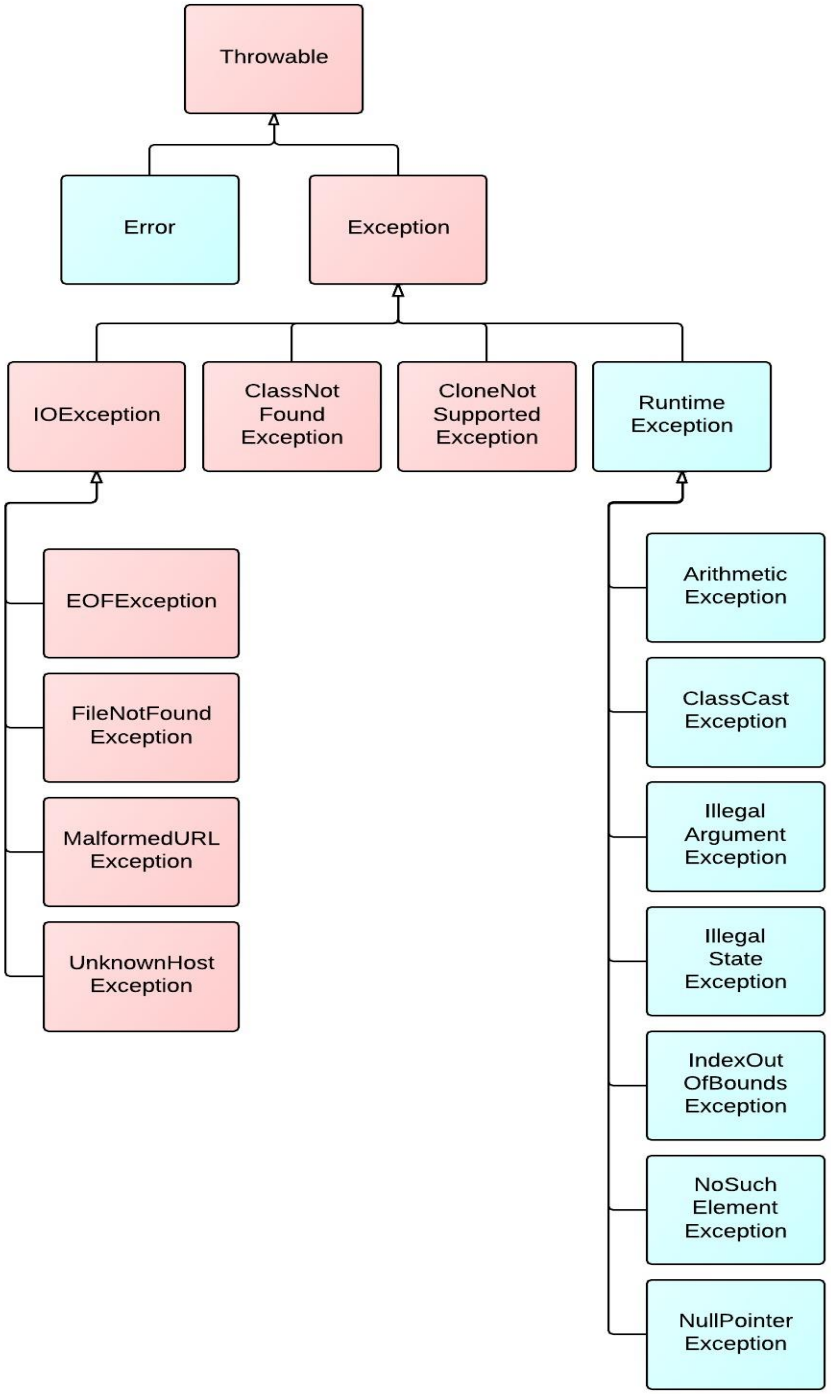
```
catch (IOException e) {
    ...
}
```

Χειρισμός όλων των I/O σφαλμάτων ανεξαρτήτως του ειδικού τους τύπου

Java Exception Hierarchy

Error	Ασυνήθιστες καταστάσεις που δεν προκαλούνται από λάθη του κώδικα αλλά γενικότερης φύσης προβλήματα που μπορεί να συμβούν (π.χ. JVM running out of memory). Δε μπορούμε να επανακάμψουμε με κάποιο τρόπο από ένα Error και δεν χρειάζεται να γράψουμε κώδικα για να το διαχειριστούμε. Πρακτικά τα Errors δεν είναι εξαιρέσεις (δεν προέρχονται από την κλάση Exception)
Exception	Δεν αφορά κάτι που είναι αποτέλεσμα προγραμματιστικού λάθους αλλά το ότι δεν είναι διαθέσιμος κάποιος πόρος ή κάποια άλλη συνθήκη που είναι απαραίτητη για την ορθή εκτέλεση του κώδικα. Π.χ. αν ο κώδικας πρέπει να συνδεθεί με κάποια άλλη εφαρμογή ή άλλο υπολογιστή που δεν ανταποκρίνεται.




Throwable
Error
Exception
IOException
ClassNot Found Exception
CloneNot Supported Exception
Runtime Exception
EOFException
FileNotFound Exception
MalformedURL Exception
UnknownHost Exception
Arithmetic Exception
ClassCast Exception
Illegal Argument Exception
Illegal State Exception
IndexOut OfBounds Exception
NoSuch Element Exception
NullPointer Exception

Δημιουργία δικής σας exception class

```
/* You should extend RuntimeException to create an unchecked
  exception, or Exception to create a checked exception. */
class MyException extends Exception {

  /* This is the common constructor. It takes a text argument.
  */
  public MyException(String p_strMessage) {
    super(p_strMessage);
  }

  /* A default constructor is also a good idea! */
  public MyException () {
    super();
  }
```

Παραδείγματα

UserDefException2

```
import java.io.*;

public class CheckingAccount {
   private double balance;
   private int number;

   public CheckingAccount(int number) {
      this.number = number;
   }

   public void deposit(double amount) {
      balance += amount;
   }

   public void withdraw(double amount) throws
InsufficientFundsException {
      if(amount <= balance) {
         balance -= amount;
      }else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }

   public double getBalance() {
      return balance;
   }

   public int getNumber() {
      return number;
   }
}
```

```
public class BankDemo {

   public static void main(String [] args) {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);

      try {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      } catch (InsufficientFundsException e) {
         System.out.println("Sorry, but you are short $" +
e.getAmount());
         e.printStackTrace();
      }
   }
}
```

```
import java.io.*;

public class InsufficientFundsException extends Exception {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
      System.out.println("passed from IFE");
   }

   public double getAmount() {
      return amount;
   }
}
```

Παραδείγματα


```
import java.io.*;

public class CheckingAccount {
   private double balance;
   private int number;

   public CheckingAccount(int number) {
      this.number = number;
   }

   public void deposit(double amount) {
      balance += amount;
   }

   public void withdraw(double amount) throws
InsufficientFundsException {
      if(amount <= balance) {
         balance -= amount;
      }else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }

   public double getBalance() {
      return balance;
   }

   public int getNumber() {
      return number;
   }
}
```

```
public class BankDemo {

   public static void main(String [] args) {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);

      try {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      } catch (InsufficientFundsException e) {
         System.out.println("Sorry, but you are short $" + e.getAmount());
         e.printStackTrace();
      }
   }
}
```

```
import java.io.*;

public class InsufficientFundsException extends Exception {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
      System.out.println("passed from IFE");
   }

   public double getAmount() {
      return amount;
   }
}
```

Παραδείγματα

UserDefException2

```
import java.io.*;

public class CheckingAccount {
   private double balance;
   private int number;

   public CheckingAccount(int number) {
      this.number = number;
   }

   public void deposit(double amount) {
      balance += amount;
   }

   public void withdraw(double amount) throws
InsufficientFundsException {
      if(amount <= balance) {
         balance -= amount;
      }else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }

   public double getBalance() {
      return balance;
   }

   public int getNumber() {
      return number;
   }
}
```

```
public class BankDemo {

   public static void main(String [] args) {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);

      try {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      } catch (InsufficientFundsException e) {
         System.out.println("Sorry, but you are short $" +
e.getAmount());
         e.printStackTrace();
      }
   }
}
```

```
Withdrawing $100...

Withdrawing $600...
passed from IFE
Sorry, but you are short $200.0

Can only enter input while your programming is running

InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:20)
        at BankDemo.main(BankDemo.java:12)
```

```
import java.io.*;

public class InsufficientFundsException extends Exception {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
      System.out.println("passed from IFE");
   }

   public double getAmount() {
      return amount;
   }
}
```

```
static void processUserInput() throws VowelException,     BlankException,     ExitException {
        System.out.print("Enter a character (x to exit): ");
                char ch;
        try {
            BufferedReader kbd = new BufferedReader(new InputStreamReader(System.in));
            String line = kbd.readLine();
            if(line.length() == 0)
                ch = ' ';
            else
                ch = Character.toUpperCase(line.charAt(0));
       }  catch (IOException x) {
            System.out.println("An IOException occurred.");
            return;}
        switch(ch) {
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U':
            throw new VowelException();
            case ' ':
            throw new BlankException();
            case 'X':
            throw new ExitException();
        } }}

class VowelException extends Exception {}
class BlankException extends Exception {}
class ExitException extends Exception {}
```

UserDefinedExceptionexample

```
Enter a character (x to exit): e
This is a vowel!
----------
Enter a character (x to exit):
You entered the blank character!
----------
Enter a character (x to exit): j
----------
Enter a character (x to exit): o
This is a vowel!
----------
Enter a character (x to exit): z
----------
Enter a character (x to exit): x
You asked to exit...
End of execution.
```

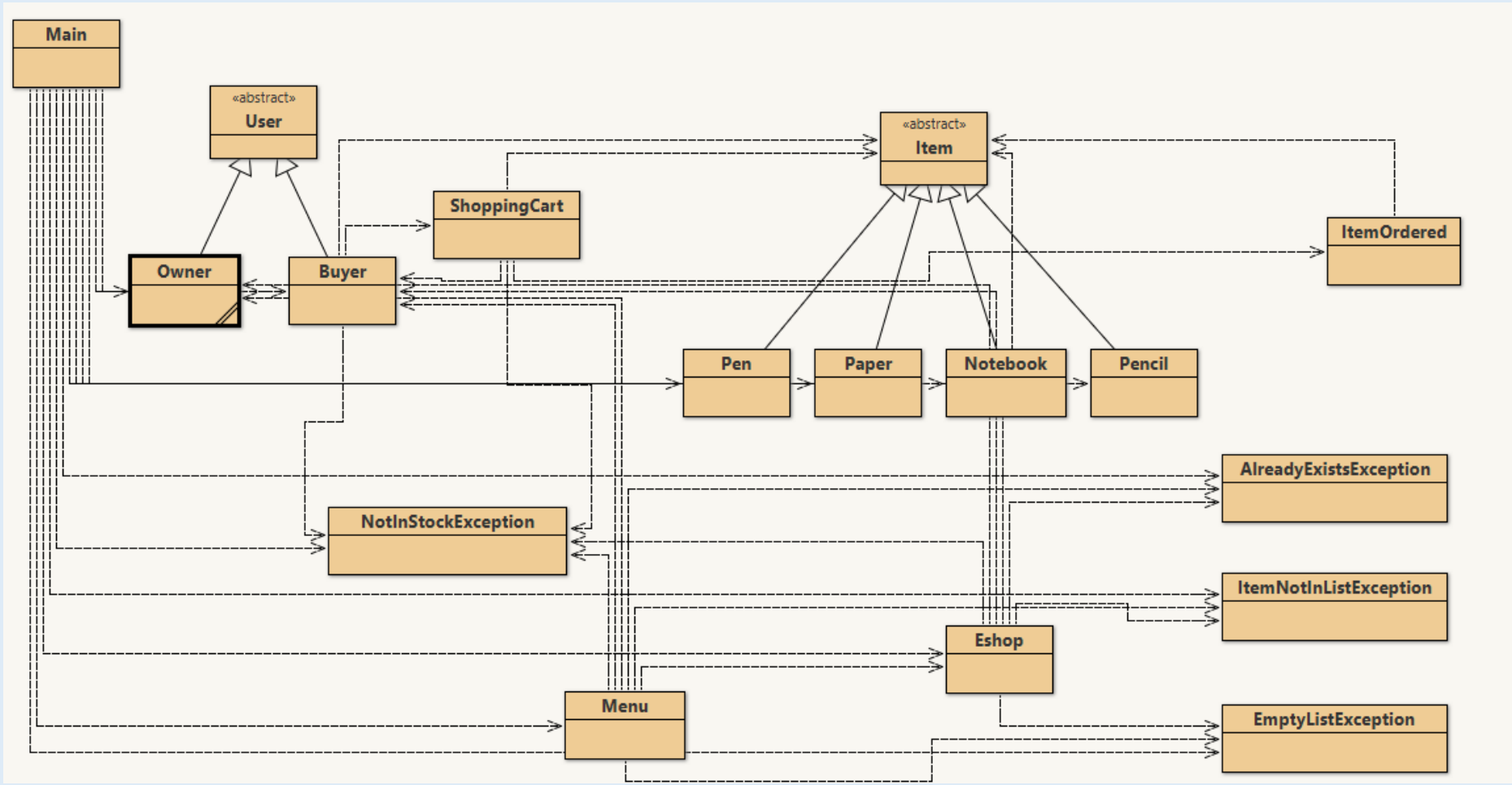
An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters

```
import java.io.*;
public class UserDefinedExceptionExample {
   public static void main(String[] args){
      boolean finished = false;
      do {
         try{
            processUserInput();
         } catch (VowelException x) {
            System.out.println("This is a vowel!");
         } catch (BlankException x) {
            System.out.println("You entered the blank character!");
         } catch (ExitException x) {
            System.out.println("You asked to exit...");
            finished = true;
            System.out.println("End of execution.");
            System.exit(0);   // finally clause not executed
         } finally {
            System.out.println("----------");
         }
      } while (!finished);
   }
```

UserDefinedExceptionexample

```
Enter a character (x to exit): e
This is a vowel!
----------
Enter a character (x to exit):
You entered the blank character!
----------
Enter a character (x to exit): j
----------
Enter a character (x to exit): o
This is a vowel!
----------
Enter a character (x to exit): z
----------
Enter a character (x to exit): x
You asked to exit...
End of execution.
```

Μια εποπτική εικόνα project με exceptions



```
public void addItem(Item i,int q)throws AlreadyExistsException{

      boolean exists = false;

      for(Item w: itemList){
      if(i.getName().equals(w.getName()))
       exists = true;
      }

      if(exists) throw new AlreadyExistsException("Υπάρχει ήδη
στην λίστα"); // new custom exception
      else{
          itemList.add(i);
          i.setStock(q);
      }

   }
```

```
public void updateItemStock(Item i,int q)throws
ItemNotInListException{
       boolean exists = false;

       for(Item w: itemList){
       if(i.getName().equals(w.getName()))
        exists = true;
      }

      if(exists)i.setStock(q);
      else throw new ItemNotInListException("Το αντικείμενο δεν
βρέθηκε"); // new custom exception

   }
```

```
public class AlreadyExistsException extends Exception{

  public AlreadyExistsException(String message){
     super(message);
  }

  public String toString(){
     return "Το αντικείμενο υπάρχει ήδη στη λίστα";
  }


}
```

```
public class ItemNotInListException extends Exception{

  public ItemNotInListException(String message){
       super(message);
  }

  public String toString(){
         return "Το αντικείμενο δεν υπάρχει στην λίστα -
ItemNotInListException";
  }
```

```
…… k = scan.nextInt();
        try{

            this.updateItemStock(itemList.get(x),k);

        }catch(ItemNotInListException inile){System.out.println(inile);}

     }…..
```