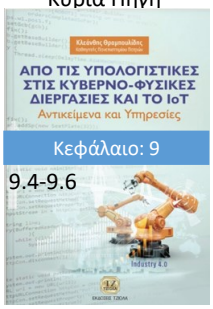
Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)

(CEID_NNY106)

Διαχείριση Εξαιρέσεων

Exception Handling - Introduction

Κύρια Πηγή



Exception:
an instance or case not conforming to the general rule.

Java

High-level programming language

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Wikipedia

Designed by: James Gosling

First appeared: May 23, 1995; 27 years ago

Paradigm: Multi-paradigm: generic, object-oriented (class-based), functional, imperative, reflective, concurrent

Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
https://sites.google.com/site/thramboulidiskleanthis/

Χειρισμός Εξαιρέσεων στην Java

- **Η Java προσφέρει μια κομψή λύση** στο πρόβλημα της διαχείρισης εξαιρέσεων:
  - **Exception handling mechanism**
- Ο μηχανισμός αυτός σας επιτρέπει να
  - γράψετε την κύρια ροή του κώδικα σας
  - να ασχοληθείτε με τις εξαιρετικές περιπτώσεις αργότερα και σε 'όποιο σημείο' θέλετε.

Οργάνωση Διάλεξης

- **Ο χειρισμός επιβάλλεται από το σύστημα**
- Η κατασκευή try/catch/finally
  - Try-with-resources
- Τύποι και αντικείμενα εξαίρεσης
- Τύποι εξαιρέσεων στην Java
- Classical error handling vs. Exception Handling
- Goody's revisited
- *Χειρισμός εξαιρέσεων στην εκπαιδευτική διαδικασία*

Ο χειρισμός εξαιρέσεων επιβάλλεται από το σύστημα

```
import java.io.*;

public class ExceptionTest0 {
   public static void main(String [] args)  {
      int num;
      num = getNumber();
      System.out.println("Το διπλάσιο του αριθμού είναι :" + 2 * num);
       }

   public static int getNumber(){
     String line;
     BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
     System.out.println("Δώσε ένα αριθμό:");
     line = br.readLine();
     return Integer.parseInt(line);
     }
  }
```

Exception java.io.IOException **must be caught**, or it **must be declared** in the throws clause of this method.
line = br.readline( );
^

Μέθοδοι εγείρουν εξαιρέσεις

```
public class BufferedReader extends Reader {
    …
    public String readLine() throws IOException
    ….
    }
```

```
public class Integer extends Number {
    ….
    public static int parseInt(String s) throws NumberFormatException
    …
    }
```

- Οι εξαιρέσεις είναι **τμήματα του public interface** της μεθόδου
- πρέπει να δηλωθούν στην υπογραφή της μεθόδου
- οι μέθοδοι ενημερώνουν τους καλούντες για τις εξαιρέσεις που μπορεί να συμβούν ώστε οι καλούντες να αποφασίσουν συνειδητά τι να κάνουν

Εξαιρέσεις της κλάσης Stack

public Object pop()

Removes the object at the top of this stack and returns that object as the value of this function.

Returns:

The object at the top of this stack (the last item of the Vector object).

***Throws*:**

EmptyStackException - if this stack is empty.

public Object peek()

Looks at the object at the top of this stack without removing it from the stack.

Returns:

the object at the top of this stack (the last item of the Vector object).

Throws: **EmptyStackException - if this stack is empty.**

public class EmptyStackException extends RuntimeException

1. Σύλληψη εξαίρεσης

```
public class ExceptionTest1 {
  ….
  public static int getNumber(){
    String line;
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    System.out.println("Δώσε ένα αριθμό:");
    try {
          line = br.readLine();                 // readLine throws IOException
          return Integer.parseInt(line);        //parseInt throws NumberFormatException
          }
    catch(IOException e) {
          e.printStackTrace();
          }
    catch(NumberFormatException e) {
          System.out.println(e);
          }
    return 0;
    }
  }
```


2. Μετάδοση εξαίρεσης

```
public class ExceptionTest2 {
  public static void main(String [] args)  {
    try{
          int num = getNumber();  // throws Exception
          System.out.println("Το διπλάσιο του αριθμού είναι :" + 2 * num);
          }
    catch(IOException e) {  System.out.println(e); }
    catch(NumberFormatException e) { System.out.println(e);}
    }
  public static int getNumber() throws Exception {
    String line;
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    System.out.println("Δώσε ένα αριθμό:");
    line = br.readLine();               // readLine throws IOException
    return Integer.parseInt(line);   //parseInt throws NumberFormatException
    }
  }
```

Οργάνωση Διάλεξης

- Ο χειρισμός επιβάλλεται από το σύστημα
- **Η κατασκευή try/catch/finally**
  - **Try-with-resources**
- Τύποι και αντικείμενα εξαίρεσης
- Τύποι εξαιρέσεων στην Java
- Classical error handling vs. Exception Handling
- Goody's revisited
- *Χειρισμός εξαιρέσεων στην εκπαιδευτική διαδικασία*

Η κατασκευή try/catch/finally

```
try {
   // κώδικας που μπορεί να εγείρει εξαίρεση
   }
catch (SomeException e1) {
   // κώδικας που χειρίζεται μια εξαίρεση e1 που είναι στιγμιότυπο της  κλάσης SomeException ή απογόνου της.
   }
catch (AnotherException e2) {
   //κώδικας που χειρίζεται μια εξαίρεση e2 που είναι στιγμιότυπο της κλάσης AnotherException ή απογόνου της.
   }
finally {
   /* κώδικας που εκτελείται πάντα, ανεξάρτητα από τον τρόπο με τον οποίο  ο έλεγχος βγήκε από το try μπλοκ : λόγω κανονικού τερματισμού, εξ αιτίας μιας πρότασης break, continue ή return ή λόγω έγερσης εξαίρεσης η οποία είτε συνελήφθη από ένα catch μπλοκ είτε όχι  */
   }
```

error handling με εξαιρέσεις – (read_file)

```
read_file {
    try {
        openTheFile;
        determine_file_length();
        allocate_that_memory();
        read_file_to_memory();
        closeTheFile;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;    }
    } catch (readFailed) { doSomething;}
      catch (fileCloseFailed) {    doSomething;      }
}
```

Try block

Χειριστής σφάλματος

Κατάλληλος χειριστής

- Ο χειριστής είναι κατάλληλος όταν
  - ο τύπος της εξαίρεσης που προκλήθηκε είναι
    - **ίδιος** με τον τύπο του χειριστή.
    - **απόγονος** (subclass) του τύπου του χειριστή.
- ο χειριστής που επιλέγεται λέμε ότι συλλαμβάνει (catches) την εξαίρεση.

Σύλληψη εξαίρεσης

- η εξαίρεση συλλαμβάνεται από
  - **τη μέθοδο που τη προκάλεσε**
    αν περιέχει κατάλληλο χειριστή ή
  - **τις καλούσες μεθόδους**
    - *Η μέθοδος την διαδίδει στις καλούσες μεθόδους* (δήλωση throws στην υπογραφή της)
    - *Ο σωρός κλήσεων ερευνάται αντίστροφα* από την μέθοδο που συνέβη το σφάλμα μέχρι να βρεθεί κατάλληλος χειριστής
    - η εξαίρεση διαδίδεται μέχρι και την main()

αν η main() διαδώσει την εξαίρεση…
το σύστημα τυπώνει μήνυμα λάθους, stack trace και τερματίζει

```
Exception in thread "main" java.lang.NullPointerException
        at Calc.printStack(Calc.java:20)
        at Calc.main(Calc.java:24)
```

Τυπική εκτύπωση της printStackTrace

```
class MyClass {
        public static void main(String[] args) {
            crunch(null);
        System.out.println("bye");
           }
        static void crunch(int[] a) {
            mash(a);
        }
        static void mash(int[] b) {
            System.out.println(b[0]);
        }
    }
```

```
java.lang.NullPointerException
    at MyClass.mash(MyClass.java:9)
    at MyClass.crunch(MyClass.java:6)
    at MyClass.main(MyClass.java:3)
```


Finally block

- **is guaranteed to be executed**, if any portion of the try block is executed
- generally used to clean up
  - close files, release resources
- in case of exception, control transfers
  - to catch block (if any) and then to the finally block,
  - first to finally block (if there is no local catch block) and then propagates.

- Λόγοι εισόδου σε finally μπλοκ
  - το try μπλοκ ολοκληρώθηκε κανονικά
  - εκτελέστηκε μια πρόταση μεταφοράς ελέγχου π.χ. return, break, continue
  - δημιουργήθηκε ένα exception
- **ο λόγος εισόδου διατηρείται για μετά το τέλος του finally**
  - **αν όμως** το finally δημιουργήσει άλλη αιτία για έξοδο (control statement, new exception) αυτή αντικαθιστά την παλιά

Παράδειγμα - writeList

```
public static void writeList() throws IOException {
    PrintWriter out = null;

    try {
      out = new PrintWriter(new FileWriter("OutFile.txt"));
      for (int i = 0; i < list.size() ; i++) {
          out.println("Value at: " + i + " = " + list.get(i));
          }
    } finally {
      if (out != null)
        out.close();
      }
}
```

Δεν υπάρχει τρόπος να φύγεις από ένα try μπλοκ χωρίς να εκτελεστεί το finally μπλοκ

προσέγγιση της Java - παρατηρήσεις

```
read_file {
    try {
        openTheFile;
        determine_file_length();
        allocate_that_memory();
        read_file_to_memory();
        closeTheFile;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

- δεν ελαττώνεται ο κόπος **ανίχνευσης, αναφοράς** και **χειρισμού** σφαλμάτων.
- **διαχωρίζεται φυσική ροή προγράμματος από λεπτομέρειες χειρισμού σφαλμάτων**
- αρχικές 7 γραμμές (bold) αυξήθηκαν σε 18

250% αύξηση!

προσέγγιση της Java - πλεονεκτήματα

```
read_file {
    try {
        openTheFile;
        determine_file_length();
        allocate_that_memory();
        read_file_to_memory();
        closeTheFile;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

- κώδικας ανίχνευσης, αναφοράς και χειρισμού
  - δεν κρύβει τις αρχικές 7 γραμμές σε λεπτομέρεια που προκαλεί σύγχυση

λογική ροή του κώδικα διατηρείται

εύκολο να διαπιστώσει κανείς ότι ο κώδικας κάνει σωστή δουλειά

»πχ κλείνει το αρχείο αν δεν μπορέσει να δεσμεύσει μνήμη;

- αν ο προγραμματιστής "λύσει" το πρόβλημα αγνοώντας το
  - τα λάθη θα αναφερθούν από το σύστημα εκτέλεσης ως unhandled exceptions

Java version vs C version

```
read_file {
    try {
        openTheFile;
        determine_file_length();
        allocate_that_memory();
        read_file_to_memory();
        closeTheFile;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

```
errorCodeType read_file {
    errorCode = 0;
    openTheFile;
    if (theFileIsOpen) {
        determine_file_length();
        if (gotTheFileLength) {
            allocate_that_memory();
            if (gotEnoughMemory) {
                read_file_to_memory();
                if (readFailed)
                    errorCode = -1;
            }
            else
                errorCode = -2;
        } else {
            errorCode = -3;
        }
    closeTheFile;
    if (theFileDidntClose &&
        errorCode == 0)
            errorCode = -4;
    else
        errorCode = errorCode and -4;
}
else {
    errorCode = -5;
}
return errorCode;
}
```

Try-with-resources in Java 7 (writeList example)

- Η διαχείριση πόρων που πρέπει άμεσα να επιστραφούν στο σύστημα είναι λίγο σύνθετη πριν την Java 7.

```
public static void writeList2() throws IOException {
   try(PrintWriter out = new PrintWriter(new FileWriter ("OutFile.txt"))) {
         for (int i = 0; i < list.size() ; i++) {
               System.out.println(list.get(i));
         }
   }
}
```

When the try block finishes the PrintWriter will be closed automatically.
This is possible because PrintWriter implements the Java interface java.lang.**AutoCloseable**.
All classes implementing this interface can be used inside the try-with-resources construct, i.e., **try()**

try-with-resource statement

- is a try statement that declares one or more resources.
  - A *resource* is an object that must be closed after the program is finished with it.
- The try-with-resources statement ensures that each resource is closed at the end of the statement.
  - Any object that implements *java.lang.AutoCloseable*, which includes all objects which implement *java.io.Closeable*, can be used as a resource.

Οργάνωση Διάλεξης

- Ο χειρισμός επιβάλλεται από το σύστημα
- Η κατασκευή try/catch/finally
- **Τύποι και αντικείμενα εξαίρεσης**
- Τύποι εξαιρέσεων στην Java
- Classical error handling vs. Exception Handling
- Goody's revisited
- *Χειρισμός εξαιρέσεων στην εκπαιδευτική διαδικασία*

Τύποι και αντικείμενα εξαίρεσης

Αττική Οδός
Κέντρο Λειτουργίας και Συντήρησης

Traffic-jam

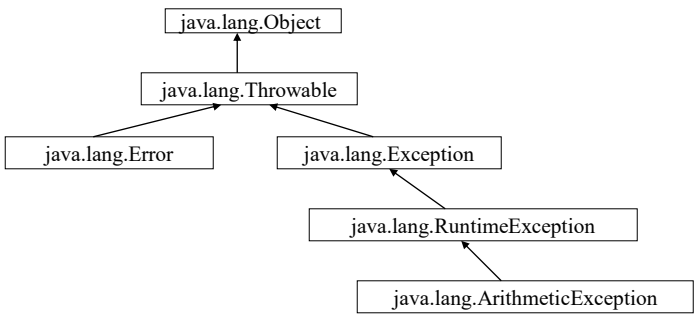
Η περίπολος συμπληρώνει μια αναφορά και την μεταδίδει στο κέντρο λειτουργίας και συντήρησης

Δήλωση Τύπου Εξαίρεσης

```
public class TrafficJamException extends RoadException {
    public TraficJamException() {
          super("traficJam");
          //String HighwayName;
          //int km;
          //int length;
          }
    public trafficJamReport(){   }
     }
```

Τύποι Exceptions

- Κάθε Exception object στην Java πρέπει να κληρονομεί
  - την κλάση **Throwable** ή
  - μια από τις απογόνους της.
- Από σύμβαση οι νέοι τύποι exception ορίζονται να κληρονομούν την κλάση **Exception** παρά την Throwable



Μη ελεγχόμενες (unchecked) exceptions

- Είναι οι **`ArithmeticException`**, **`RuntimeException`** και όλες οι υποκλάσεις της
- ο compiler δεν ελέγχει τις ***unchecked exceptions.***
- **είναι ευθύνη του προγραμματιστή** να κάνει τους απαραίτητους ελέγχους για να αποτρέψει την εμφάνιση τους.
- Το Eclipse δεν μας προειδοποιεί για Unhandled Exception π.χ. για την **`pop()`** η οποία εγείρει **`EmptyStackException`** (υποκλάση της `RuntimeException`).

Δημιουργία – Δήλωση τύπου Exception

- ένας νέος τύπος εξαίρεσης πρέπει να δημιουργείται όταν ο προγραμματιστής θέλει να χειριστεί ένα τύπο λάθους ξεχωριστά από τους άλλους

```
class MidLevelException extends Exception {
        MidLevelException(String message)  { super(message); }
    }

class LowLevelException extends Exception { ....      }
```

```
public class DivideByZeroException extends ArithmeticException {
  public DivideByZeroException() {
      super("Attempted to divide by zero");
      }
  }
```

αντικείμενα εξαίρεσης

- **Είναι**
  - στιγμιότυπα κλάσης που περιγράφει τη δομή και τη συμπεριφορά τους
- **περιέχουν** πληροφορία σχετικά με την εξαίρεση
  - τον τύπο της,
  - επιπρόσθετα εξατομικευμένα (custom) πεδία,
  - την κατάσταση προγράμματος όταν το σφάλμα συνέβει.
- **Μεταδίδονται**
  - με την λέξη-κλειδί **throw**

Έγερση εξαίρεσης - throw statement

- Μια εξαίρεση εγείρεται:
  - Καλώντας μια μέθοδο που εγείρει εξαίρεση, ή
  - Χρησιμοποιώντας την πρόταση **throw**
- Η πρόταση **throw**
  - χρησιμοποιείται για την έγερση εξαίρεσης
  - δέχεται σαν παράμετρο ένα αντικείμενο το οποίο και κάνει throws

```
if(expression)
      throw new MidLevelException(e);
```

σύστημα αναλαμβάνει σε κάθε περίπτωση να βρει τον κατάλληλο κώδικα για να χειριστεί το σφάλμα (exception handling).

Παράδειγμα Έγερσης εξαίρεσης

```
public int quotient (int numerator, int denominator) {
        return numerator / denominator;
        }
```

```
public int quotient (int numerator, int denominator)
                                throws DivideByZeroException {

        if(denominator = = 0)
                throw new DivideByZeroException();

        return numerator / denominator;
        }
```


EmptyStack Exception in Stack

```
double pop(void) {
    if(sp>0)
        return val[--sp];
    else {
        printf("error: stack empty\n");
        return 0.0; }
}
```

Η pop αναγνωρίζει και χειρίζεται την εξαίρεση

Η pop αναγνωρίζει και μεταδίδει την εξαίρεση

```
double pop(void) throws EmptyStackException {
    if(sp>0)
        return val[--sp];
    else {
        printf("error: stack empty\n");
        return 0.0; }
        throw new EmptyStackException();
}
```

Δήλωση της EmptyStackException

Η εξαίρεση εγείρεται από μεθόδους της κλάσης Stack για να δηλώσουν ότι η στοίβα είναι άδεια.

```
public class EmptyStackException extends RuntimeException{
        public EmptyStackException() { //πιθανή υλοποίηση
                super("Stack is empty");
                }
    }
```


Οργάνωση Διάλεξης

- Ο χειρισμός επιβάλλεται από το σύστημα
- Η κατασκευή try/catch/finally
- Τύποι και αντικείμενα εξαίρεσης
- **Τύποι εξαιρέσεων στην Java**
- Classical error handling vs. Exception Handling
- Goody's revisited
- *Χειρισμός εξαιρέσεων στην εκπαιδευτική διαδικασία*

Τύποι εξαιρέσεων στην Java

String
StringBuffer
StringBuilder
System
Thread
ThreadGroup
ThreadLocal
Throwable
Void

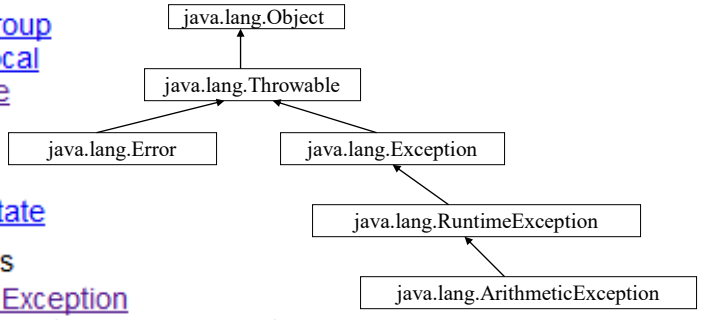
Enums

Thread.State

Exceptions

ArithmeticException
ArrayIndexOutOfBoundsException
ArrayStoreException
ClassCastException

java.lang Exceptions



Exceptions
ArithmeticException
ArrayIndexOutOfBoundsException
ArrayStoreException
ClassCastException
ClassNotFoundException
CloneNotSupportedException
EnumConstantNotPresentException
Exception
IllegalAccessException
IllegalArgumentException
IllegalMonitorStateException
IllegalStateException
IllegalThreadStateException
IndexOutOfBoundsException
InstantiationException
InterruptedException
NegativeArraySizeException
NoSuchFieldException
NoSuchMethodException
NullPointerException
NumberFormatException
RuntimeException
SecurityException
StringIndexOutOfBoundsException
TypeNotPresentException
UnsupportedOperationException

Class Throwable

java.lang
Class Throwable

java.lang.Object
└ java.lang.Throwable

- Είναι η superclass όλων των errors και exceptions στην Java,
- μόνο instances αυτής ή των απογόνων της εγείρονται από την JVM ή μπορούν να εγερθούν με την πρόταση **throw**
- Μόνο η κλάση αυτή ή απόγονος της μπορεί να είναι όρισμα ενός **catch**
- Περιέχει ένα String που χρησιμοποιείται για την αποθήκευση ενός **human-readable μηνύματος λάθους**.
  - Αυτό αρχικοποιείται από το δημιουργό.

Μέθοδοι της Throwable

- **public String getMessage()**

  Returns: the detail message string of this Throwable instance (which may be null).

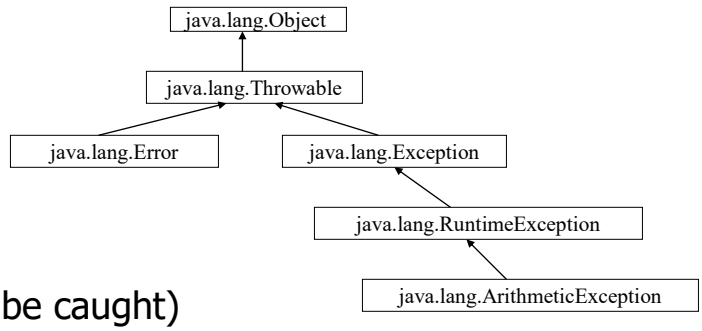
- **public String toString()**

  Returns a short description of this throwable.

- **public void printStackTrace()**

  Prints this throwable and its backtrace to the standard error stream.

Standard exception objects



- two standard subclasses
  - java.lang.Error (should not be caught)
    - indicate linkage problems related to dynamic loading
    - virtual machine problems e.g., out of memory
  - java.lang.Exception (may be caught)
    - e.g. EOFException, ArrayAccessOutOfBounds

class Exception

public class **Exception** extends Throwable

- class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

public Exception(String message)

Constructs a new exception with the specified detail message.

The detail message is saved for later retrieval by the getMessage() method of Throwable.

Οργάνωση Διάλεξης

- Ο χειρισμός επιβάλλεται από το σύστημα
- Η κατασκευή try/catch/finally
- Τύποι και αντικείμενα εξαίρεσης
- Τύποι εξαιρέσεων στην Java
- **Classical error handling vs. Exception Handling**
- Goody's revisited
- *Χειρισμός εξαιρέσεων στην εκπαιδευτική διαδικασία*

Classical error handling vs. Exception Handling

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <stdlib.h>
#include <stdio.h>

int main() {

        int fh;
        char buffer[65];
        int gotten;

        fh = open("abc.txt",O_RDONLY);
        printf ("File handle %d\n",fh);
        while (gotten = read(fh,buffer,64)) {
                buffer[gotten] = '\0';
                printf("******%s",buffer);
                }
        }
```

Error handling?

Java library

```
public FileReader(File file)
            throws FileNotFoundException

public FileWriter(String fileName)
            throws IOException
```

- Instead of returning -1 a throws clause has to be defined
- FileOpenException type has to be defined (alternatives?)
- Methods calling open should
  - Catch FileOpenException, or
  - Declare it in their throws clause

Οργάνωση Διάλεξης

- Ο χειρισμός επιβάλλεται από το σύστημα
- Η κατασκευή try/catch/finally
- Τύποι και αντικείμενα εξαίρεσης
- Τύποι εξαιρέσεων στην Java
- Classical error handling vs. Exception Handling
- **Goody's revisited**
- *Χειρισμός εξαιρέσεων στην εκπαιδευτική διαδικασία*

Goody's Exceptions (Revisited)

- Στην εκτέλεση της ενέργειας «πάρε τυρί» διαπιστώνει ότι το τυρί έχει τελειώσει.
  - Το συμβάν αυτό χαρακτηρίζεται ως **εξαιρετική περίπτωση** που όμως είναι πιθανή καθώς η Μαίρη προετοιμάζει το τοστ
- Αν η Μαίρη δεν έχει ενημερωθεί πως να την αναγνωρίσει και να την χειριστεί το σύστημα κινδυνεύει να μεταπέσει σε κατάσταση απροσδιόριστης συμπεριφοράς.

```
public double quotient (int numerator, int denominator) {

        //      if(denominator = = 0)

        return (double) numerator / denominator;
}
```


Διαχείριση Εξαίρεσης – 1ο σενάριο

- Η Μαίρη
  - έχει την γνώση να χειριστεί τη συγκεκριμένη εξαίρεση, πρέπει να εκτελέσει ένα σύνολο από ενέργειες όπως «πάρε από το ψυγείο της αποθήκης ένα κεφάλι τυρί», «κόψε το σε φέτες, », κ.λ.π. Λέμε πως
  - **αναγνωρίζει την εξαίρεση** και **την αντιμετωπίζει**. Δεν είναι απαραίτητο να ενημερώσει τον Γιώργο για την εξαιρετική αυτή περίπτωση.

```
public double quotient (int numerator, int denominator) {

        if(denominator = = 0) {
                System.out.println(....);
                return;
                }
        return (double) numerator / denominator;
        }
```

Διαχείριση Εξαίρεσης – 2ο σενάριο (1/2)

- Ο Γιώργος δεν θέλει να αναθέσει στην Μαίρη την αντιμετώπιση της εξαιρετικής αυτής περίπτωσης για διάφορους λόγους.
  - Στην περίπτωση αυτή η Μαίρη αναγνωρίζει την εξαιρετική περίπτωση δημιουργεί ένα μήνυμα όπως «δεν υπάρχει τυρί» (ένα αντικείμενο εξαίρεσης θα έλεγα) και το στέλνει στον Γιώργο που είναι ο προϊστάμενος του τμήματος. Στην περίπτωση αυτή λέμε πως
- η Μαίρη αναγνωρίζει την εξαιρετική περίπτωση, εγείρει ένα αντικείμενο εξαίρεσης (throw an exception) και την μεταδίδει (**throws**) στην διεργασία που την κάλεσε.

```
public double quotient (int numerator, int denominator) throws DivideByZeroException
        if(denominator = = 0)
                throw new DivideByZeroException();
        return (double) numerator / denominator;
        }
```


Διαχείριση Εξαίρεσης – 2ο σενάριο (2/2)

- Ο Γιώργος θα πρέπει **να συλλάβει (catch)** την εξαίρεση και να την αντιμετωπίσει κατάλληλα.

```
public class ExceptionTest1 {
  ....
  public static int getNumber(){
          String line;
          BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
          System.out.println("Δώσε ένα αριθμό:");
          try {
             line = br.readLine();              // readLine throws IOException
             return Integer.parseInt(line);     //parseInt throws NumberFormatException
             }
          catch(IOException e) { System.out.println(e); }
          catch(NumberFormatException e) {      System.out.println(e);  }
          return 0;
          }
  }
```

Διαχείριση Εξαίρεσης – 3ο σενάριο

- όπως η επιλογή No 2 με τη διαφορά πως ο Γιώργος **κάνει throws την εξαίρεση** αν δεν μπορεί να την χειρισθεί

```
public class ExceptionTest2 {
  public static void main(String [] args)  {
        ….
        }

  public static int getNumber() throws Exception {
        String line;
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        System.out.println("Δώσε ένα αριθμό:");
        line = br.readLine();                  // readLine throws IOException
        return Integer.parseInt(line);         //parseInt throws NumberFormatException
        }
  }
```


Εκπαιδευτική διαδικασία στο Πανεπιστήμιο Πατρών

Χειρισμός Εξαιρέσεων

Κλεάνθης Θραμπουλίδης
Software Engineering Group
Electrical & Computer Engineering
University of Patras, Greece

Ανάγκη για αξιόπιστο χειρισμό εξαιρέσεων

Java Επιβάλλεται από το σύστημα

Ιούλιοι 'προγραμματιστές' "λύνουν" το πρόβλημα απλά αγνοώντας το. Τα λάθη εμφανίζονται μαζί με το επόμενο crash.

C Ευθύνη του προγραμματιστή

«η τήρηση κανόνων είναι υπόθεση του 'πατριωτισμού' όλων»

http://seg.ceid.upatras.gr/thrambo/theseis/docs/20Feb2008.pps