Προγραμματισμός ΙΙ (Java)

4. Διαχείριση αρχείων

Διαχείριση Εξαιρέσεων

Περιεχόμενα

- Ρεύματα
  - Προκαθορισμένα ρεύματα εισόδου/εξόδου
- Η τάξη File - Οργάνωση αρχείων
  - Ρεύματα bytes, χαρακτήρων
- Αρχεία κειμένου
  - Ρεύματα εισόδου αρχείων
  - Ρεύματα εξόδου αρχείων
- Αρχεία binary
  - Σειριακή μεταφορά αντικειμένων
- Αρχεία άμεσης προσπέλασης
- Διαχείριση λαθών – Εξαιρέσεις

Ρεύματα

- Όταν ένα πρόγραμμα επικοινωνεί με την οθόνη, το πληκτρολόγιο, ένα αρχείο κλπ., χρησιμοποιεί ρεύματα χαρακτήρων (streams) για να στείλει ή να πάρει δεδομένα.
- Τρία προκαθορισμένα ρεύματα στη Java:
  - System.in: Διαβάζει bytes ***συνήθως*** από το πληκτρολόγιο.
  - System.out: Στέλνει bytes ***συνήθως*** στην οθόνη.
  - System.err: Στέλνει μηνύματα λάθους ***συνήθως*** στην οθόνη.
- «συνήθως». Μπορεί όμως να συνδεθεί με άλλη πηγή με τις μεθόδους setIn, setOut, setErr της System

Αρχεία Text και Binary

- Αρχεία Text: περιέχουν τα δεδομένα σε μορφή εκτυπώσιμων χαρακτήρων
  - Ένα byte για κάθε χαρακτήρα (για την κωδικοποίηση ASCII)
  - Δύο bytes για κάθε χαρακτήρα (character) (για την κωδικοποίηση Unicode, και για πολλές γλώσσες)
  - Μπορούμε να διαβάσουμε τα αρχεία από ένα "text editor"
- Αρχεία Binary: περιέχουν κωδικοποιημένα δεδομένα, όπως εντολές προς εκτέλεση ή αριθμητικά δεδομένα
  - Δεν είναι κατάλληλα για εκτύπωση
  - Μπορούν να διαβαστούν από υπολογιστές αλλά όχι από ανθρώπους
  - Είναι πιο εύκολο για ένα πρόγραμμα να τα χειριστεί.

Ρεύματα εισόδου/εξόδου για κείμενο

- Οι σημαντικότερες κλάσεις για έξοδο (σε αρχείο):
  - PrintWriter
  - FileOutputStream [ή FileWriter]
- Οι σημαντικότερες κλάσεις για είσοδο (από αρχείο):
  - BufferedReader
  - FileReader
- Η FileOutputStream και η FileReader δέχονται ως όρισμα ένα όνομα αρχείου
- Η PrintWriter και η BufferedReader έχουν χρήσιμες μεθόδους για γραφή και ανάγνωση

Προσωρινή αποθήκευση (buffering)

- Κάθε byte που διαβάζουμε από το αρχείο (ή το πληκτρολόγιο) τοποθετείται σε μια προσωρινή μνήμη. Μόλις συγκεντρωθούν «αρκετά» bytes τα χειριζόμαστε ανάλογα (π.χ. τα εμφανίζουμε στην οθόνη, τα μετατρέπουμε στον κατάλληλο τύπο δεδομένων κλπ)
- Με buffer: γραφή και ανάγνωση σε τμήματα (“chunks”)
  - Καθυστερεί ο χειρισμός ορισμένων bytes (καθώς περιμένουν να γεμίσει ο buffer) π.χ. σε ένα 16-byte buffer, μπορεί να περιμένουμε να διαβαστούν όλα τα bytes πριν χρησιμοποιήσουμε τα 4 πρώτα για έναν int ή δεν γράφουμε τον ακέραιο στο αρχείο, μέχρι να γεμίσει ο buffer, οπότε γράφουμε 4 int μαζί
  - Λιγότερος επιπλέον φόρτος πρόσβασης στο δίσκος (I/O overhead)
- Χωρίς buffer: επεξεργαζόμαστε κάθε byte μόλις έρθει στη μνήμη
  - Μικρότερη καθυστέρηση για το χειρισμό του ενός byte
  - Μεγάλος επιπλέον φόρτος γραφής και ανάγνωσης από το δίσκο

Η κλάση File

- Είναι η βασική κλάση για αρχεία και φακέλους
- Μέθοδοι της `File`
  - `exists`: ελέγχει αν υπάρχει το αρχείο
  - `canRead`: ελέγχει αν μπορούμε να διαβάσουμε από το αρχείο
  - `canWrite`: ελέγχει αν μπορούμε να γράψουμε στο αρχείο
  - `delete`: διαγράφει το αρχείο και επιστρέφει true (αν πετύχει)
  - `length`: επιστρέφει το μέγεθος του αρχείου σε bytes
  - `getName`: επιστρέφει το όνομα του αρχείου (μόνο)
  - `getPath`: επιστρέφει το πλήρες μονοπάτι του αρχείου

```
File numFile = new File(“numbers.txt”);
if (numFile.exists())
  System.out.println(numfile.length());
```

Σχετικές κλάσεις

- Η FileInputStream και η FileOutputStream έχουν κατασκευαστές που παίρνουν ως όρισμα ένα αντικείμενο File ή το όνομα του αρχείου ως String

```
PrintWriter smileyOutStream = new
  PrintWriter(new
  FileOutputStream("smiley.txt"));
File smileyFile = new File("smiley.txt");
if (smileyFile.canWrite())
  PrintWriter smileyOutStream = new
  PrintWriter(new
  FileOutputStream(smileyFile));
```

Αρχεία κειμένου

Ρεύματα για bytes (Java 1.0)

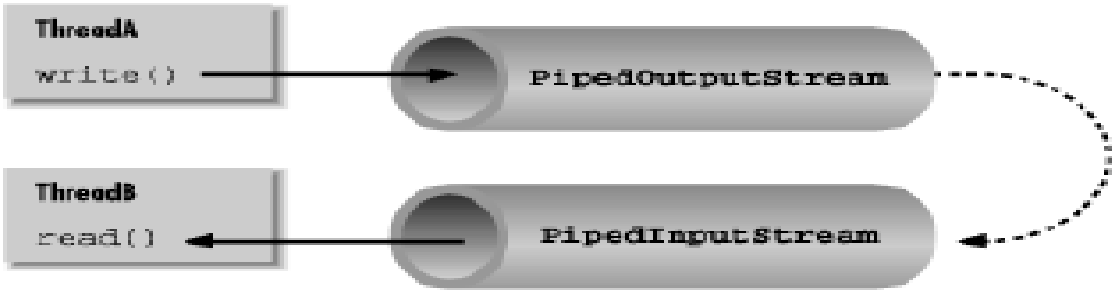
- Όλα τα ρεύματα που θα δούμε βρίσκονται στο πακέτο io.
- Βρίσκονται κάτω από τις abstract τάξεις InputStream και OutputStream οι οποίες περιέχουν τις βασικές μεθόδους για είσοδο και έξοδο bytes.
  - *read(), skip(), available(), mark(int), reset(), close()*
- Για περισσότερες λειτουργίες στα ρεύματα φτιάχτηκαν οι τάξεις FilterInputStream και FilterOutputStream.
  - Κατασκευαστής: *FilterInputStream(InputStream in)*
  - Πεδία: *in* το InputStream με το οποίο συνδέεται

Ρεύματα για bytes (2)

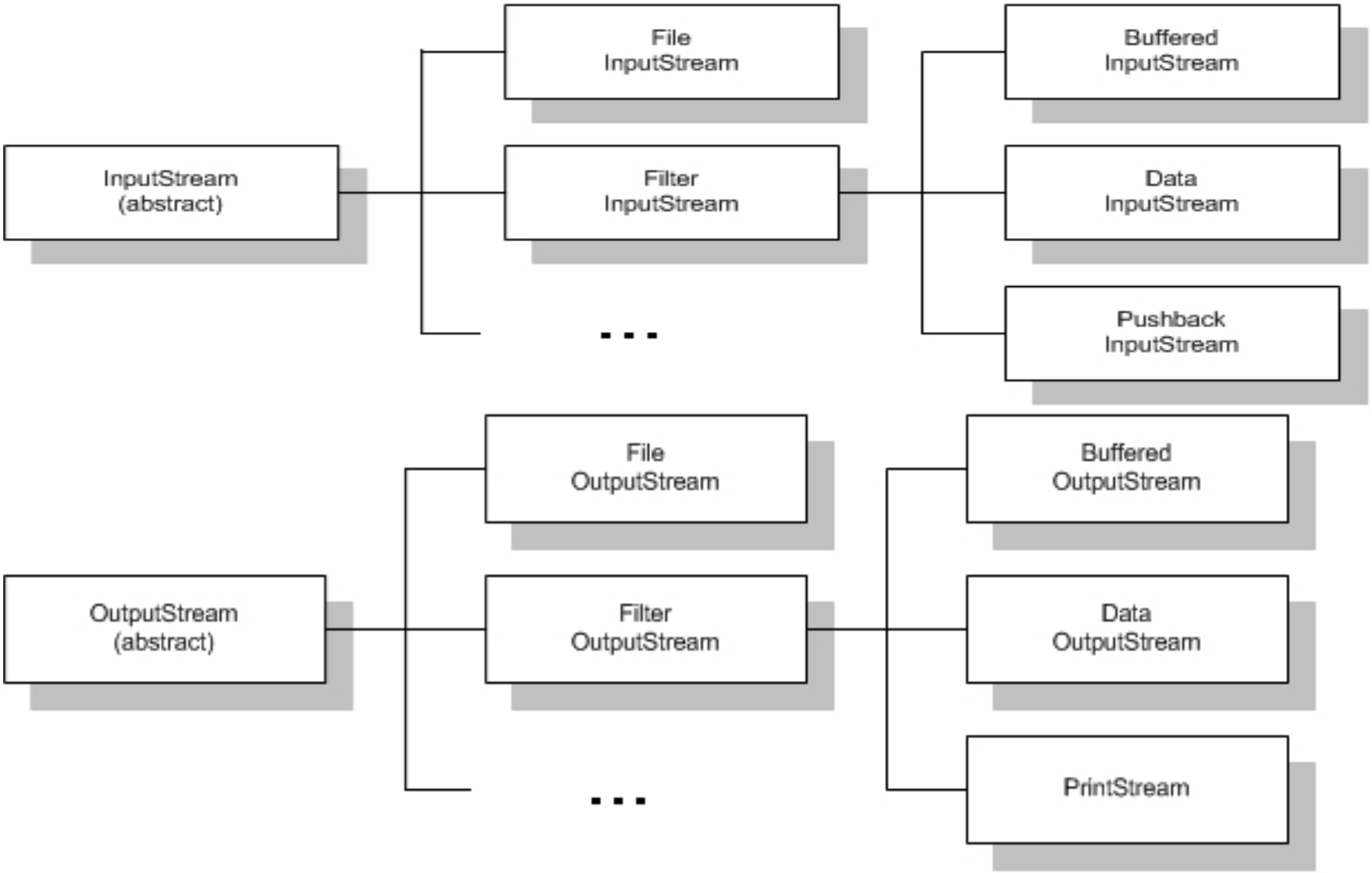
- Τάξεις που τις κληρονομούν είναι οι PrintStream (π.χ. η System.out), οι BufferedInputStream και BufferedOutputStream που έχουν κάποιο buffer,
  - Κατασκευαστής: *BufferedInputStream(InputStream in)*
  - Πεδία: *buf :* ένα byte[] που αποθηκεύει τα δεδομένα
    count: το πλήθος των bytes στο buffer
- οι DataInputStream και DataOutputStream που διαβάζουν βασικούς τύπους υλοποιώντας αντίστοιχα interfaces (DataInput και DataOutput).
  - Κατασκευαστής: *DataInputStream(InputStream in)*
  - Μέθοδοι: *readFloat*(), διαβάζει 4 bytes και επιστρέφει float

Ρεύματα για bytes (3)

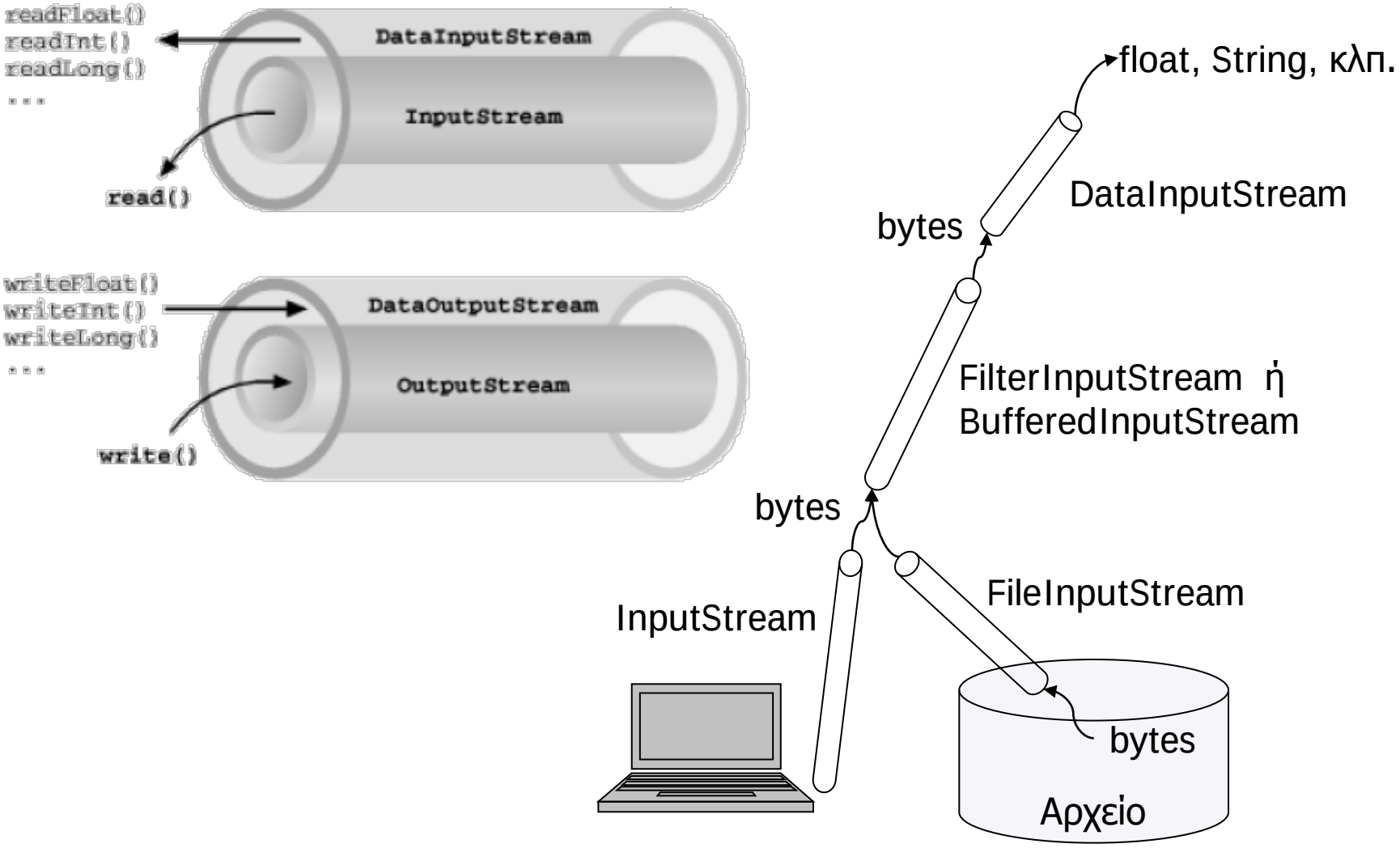
- Για ανταλλαγή δεδομένων όταν έχουμε πολλά threads υπάρχουν οι PipedInputStream / PipedOutputStream
  - Κατασκευαστής: PipedInputStream()
    PipedOutputStream(PipedInputStream src)
  - Μέθοδοι: connect(PipedInputStream snk)
    write(byte[] b, int off, int len)
- Για αρχεία υπάρχουν οι FileInputStream και FileOutputStream.
  - Κατασκευαστής: *FileInputStream(String filename)*
    throws FileNotFoundException



Ιεραρχία

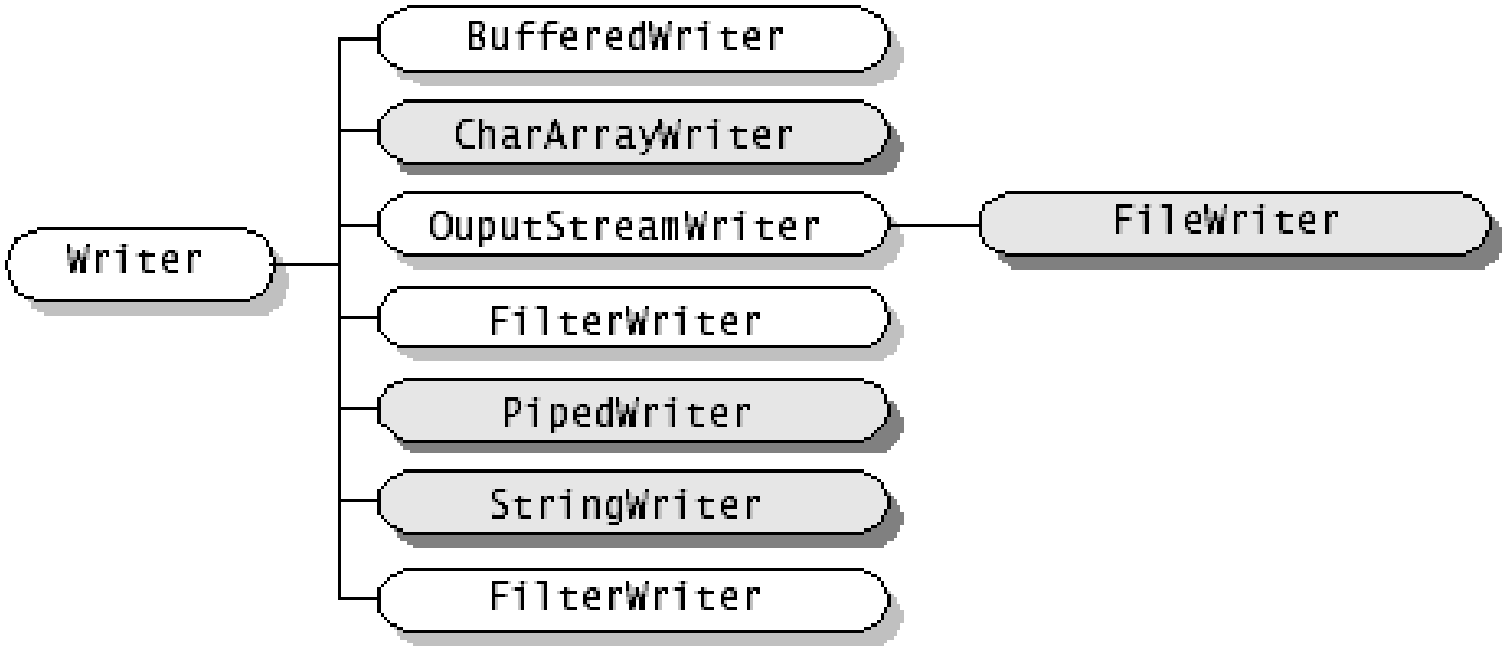

InputStream
(abstract)
File
InputStream
Filter
InputStream
...
Buffered
InputStream
Data
InputStream
Pushback
InputStream
OutputStream
(abstract)
File
OutputStream
Filter
OutputStream
...
Buffered
OutputStream
Data
OutputStream
PrintStream

Συνδυασμός ρευμάτων bytes



Ρεύματα για χαρακτήρες (Java 1.1)

- Για τη διαχείριση ρευμάτων 16-bit (Unicode) χαρακτήρων υπάρχουν οι abstract τάξεις Reader και Writer με αντίστοιχες ιεραρχίες



Βασικές μέθοδοι

- Οι Reader/Writer έχουν μεθόδους για χαρακτήρες

  int read()

  // ο ακέραιος αντιστοιχεί σε κάποιο char με δεδομένο charset

  // -1 όταν το ρεύμα είναι άδειο (π.χ. EOF)

  int read(char cbuf[])

  int read(char cbuf[], int offset, int length)

  int write(int c)

- Οι InputStream/OutputStream αντίστοιχα για bytes

  π.χ. **int write(byte b[], int offset, int length)**

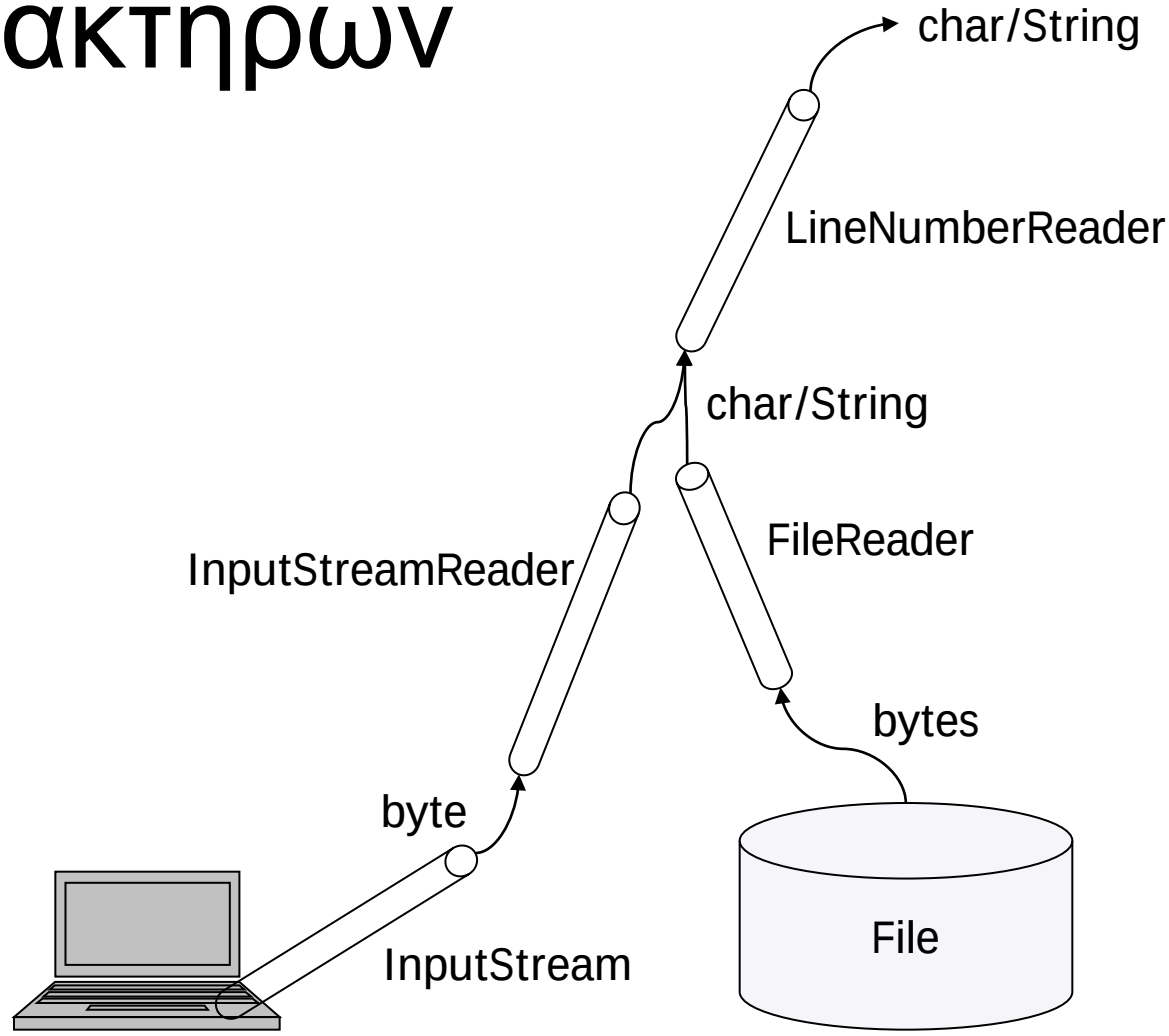
- Οι InputStreamReader και OutpuStreamWriter μπορούν να χρησιμοποιηθούν ως γέφυρες
  - Κατασκευαστής: *InputStreamReader(InputStream in)*

Ρεύματα για χαρακτήρες (2)

- LineNumberReader
  - Κατασκευαστής: *LineNumberReader(Reader in)*
  - *Μέθοδοι: readLine()* : διαβάζει μια γραμμή σε String
    getLineNumber(): επιστρέφει τον αριθμό γραμμής
- *PrintWriter*
  - Κατασκευαστές: *PrintWriter(File file)*
    PrintWriter(OutputStream out)
    PrintWriter(String fileName)
  - *Μέθοδοι: print(float f), print(String s), println(Object o)*
    write(String s)

ΠΡΟΣΟΧΗ: Όταν δε χρειαζόμαστε πλέον ένα stream, ένα reader ή writer τον κλείνουμε με τη μέθοδο ***close()***

Συνδυασμός ρευμάτων χαρακτήρων



Διάβασμα από αρχεία

- Η τάξη File (στο πακέτο java.io.*) αντιπροσωπεύει:
  - Ένα αρχείο:
    File arxeio = new File("c:\\1.txt");
  - Ένα κατάλογο:
    File katalogos = new File(".");
    String path=katalogos.getAbsolutePath();
    File[] files=katalogos.listFiles();
- Η τάξη FileReader (FileWriter) δημιουργεί ένα ρεύμα εισόδου (εξόδου) από αρχείο:
  - FileReader arxeio = new FileReader("c:\\1.txt"));

Ανάγνωση - Εγγραφή

- Ανάγνωση: Όπως και με το πληκτρολόγιο συνδέουμε το πρόγραμμά μας με το αρχείο με έναν BufferedReader

```
BufferedReader eisodos = new BufferedReader(new
                                FileReader("c:\\1.txt"));
```

- Εγγραφή: Συνδέουμε το πρόγραμμά μας με το αρχείο με ένα BufferedWriter

```
PrintWriter exodos =new PrintWriter(new BufferedWriter(new
                                    FileWriter(“c:\\copy.txt")));
String s;
while((s = eisodos.readLine()) != null )
  exodos.println(lineCount++   + ": " + s);
exodos.close();
```

Παράδειγμα - Αντιγραφή

```
import java.io.*;
class test{
  public static void main(String args[]){
      try{
        BufferedReader eisodos = new BufferedReader(new
                              FileReader("c:\\autoexec.bat"));
        PrintWriter exodos =new PrintWriter(new BufferedWriter(new
              FileWriter(“c:\\copy.txt")));
        String s;
        while((s = eisodos.readLine()) != null )
              exodos.println(s);
        exodos.close();
      }
      catch (Exception ex){ System.out.println(ex);}
  }
}
```

Binary αρχεία

Σειριακή μεταφορά αντικειμένων

- Όταν θέλουμε να στείλουμε ολόκληρα αντικείμενα σε ένα ρεύμα χρησιμοποιούμε τις τάξεις *ObjectInputStream και ObjectOutputStream*
- Απεικονίζουμε το αντικείμενο με σειριακή μορφή ώστε να μπορούμε να το ανακτήσουμε (serialization)
  - Κατασκευαστής: ObjectOutputStream(OutputStream out)
  - Μέθοδοι: writeInt(int val), writeObject(Object o), flush()
  - Κατασκευαστής: ObjectInputStream(InputStream in)
  - Μέθοδοι: int readInt(), Object writeObject(), flush()
- Η κλάση του αντικειμένου πρέπει να υλοποιεί το Serializable interface

Παράδειγμα

```
try{
    FileOutputStream out = new FileOutputStream(“output.dat");
    ObjectOutputStream s = new ObjectOutputStream(out);
    s.writeObject("Today");
    s.writeObject(new Date());
    s.flush();
    s.close();
    FileInputStream in = new FileInputStream("output.dat");
    ObjectInputStream t=new ObjectInputStream(in);
    String today = (String)t.readObject();
    Date date = (Date)t.readObject();
    t.close();
    System.out.println(today+”\n”+date);
}
catch (Exception e){
    e.printStackTrace();
}
```

`Serializable` Interface

- Ένα αντικείμενο είναι serializable αν η κλάση του υλοποιεί το java.io.Serializable interface.
- Το interface δεν έχει επιπλέον μεθόδους για να υλοποιήσουμε. Είναι marker interface.
- Με τη δήλωση επιτρέπουμε στην Java να αυτοματοποιήσει το μηχανισμό αποθήκευσης των αντικειμένων από/σε αρχεία
- Μπορούμε επίσης να καθορίζουμε τι θα αποθηκεύεται και τι όχι

Προσοχή

- Τι γίνεται αν το αντικείμενο περιέχει γνωρίσματα μη σειριοποιήσιμα;
  - Δεν σειριοποιείται.
- Όμοια και ένας πίνακας αντικειμένων
- Μπορούμε να σειριοποιήσουμε τα υπόλοιπα γνωρίσματα δηλώνονται ως transient αυτά που θα αγνοηθούν

```
public class Foo implements java.io.Serializable {
  private int v1; // σειριοποιείται
  private static double v2;
  private transient A v3 = new A();
}
```

Αρχεία τυχαίας πρόσβασης

- Η τάξη RandomAccessFile δημιουργεί αρχεία στα οποία η ανάγνωση και γραφή δε γίνεται σειριακά αλλά σε οποιαδήποτε θέση.
  - Κατασκευαστής:

    RandomAccessFile(String name, String mode)

    RandomAccessFile(File file, String mode)

    mode: “r” μόνο για ανάγνωση, “rw” για ανάγνωση/εγγραφή
  - Μέθοδοι:

    skipBytes(int n) //προχωρά το δείκτη κατά n bytes

    seek(long pos) //πάει το δείκτη στη θέση pos του αρχείου

    long getFilePointer() //επιστρέφει τη θέση του δείκτη

Πώς ελέγχω το τέλος του αρχείου;

- Βάζω έναν ειδικό χαρακτήρα και σταματώ μόλις τον διαβάσω
- Ψάχνω για έναν ειδικό χαρακτήρα στο τέλος του αρχείου (τα text αρχεία έχουν τέτοιο χαρακτήρα)
- Χρησιμοποιώ την κλάση Scanner

  ```
  while (inFile.hasNext())
  while (inFile.next()!=null)
  ```

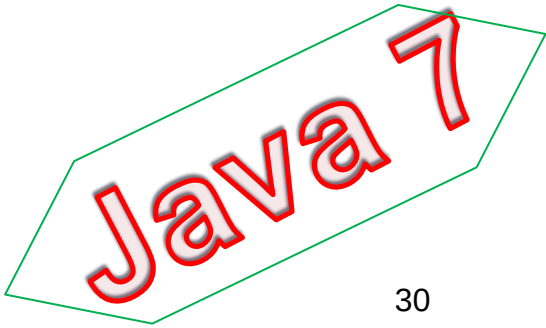
- Πυροδοτώ μια εξαίρεση τέλους-αρχείου και την ανιχνεύω στον κώδικά μου

Java 7

Νέο πακέτο java.nio.file

- Περιέχει κλάσεις και interfaces όπως οι Path, Paths, Files, DirectoryStream, WatchService κλπ.
- Παράδειγμα: αντιγραφή αρχείου:

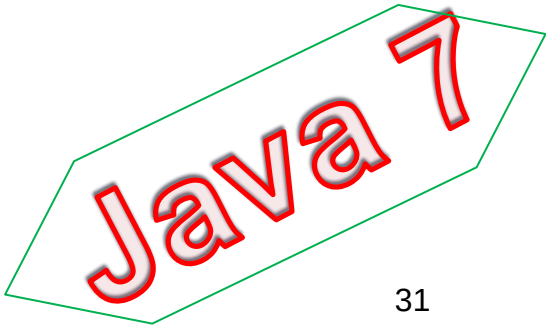
```
Path src = Paths.get("/home/fred/readme.txt");
Path dst = Paths.get("/home/fred/copy_readme.txt");
Files.copy(src, dst,
           StandardCopyOption.COPY_ATTRIBUTES,
           StandardCopyOption.REPLACE_EXISTING);
```



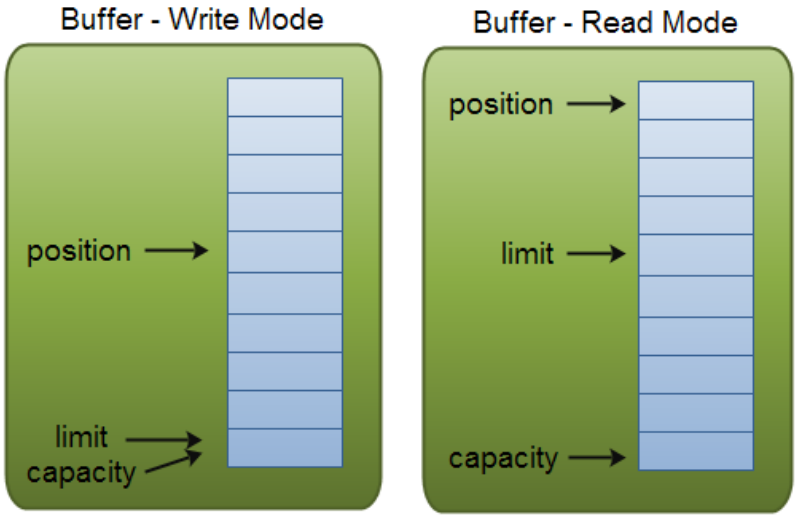
http://cs.brown.edu/courses/cs161/papers/j-nio-ltr.pdf

java.nio.channels

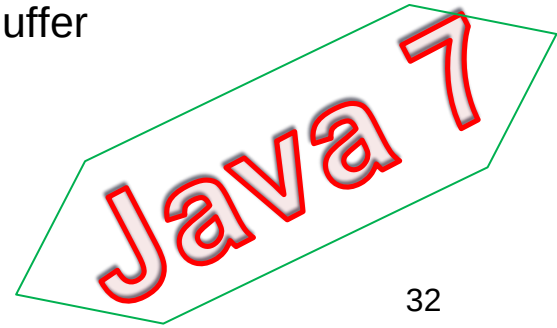
- java.nio.Channel → Stream
  - Read & Write → Read or write
  - Read & Write asynchronously
  - Πάντα επικοινωνούμε μέσω ενός Buffer αντικειμένου
- Implementations
  - FileChannel για αρχεία
  - DatagramChannel για συνδέσεις UDP.
  - SocketChannel για συνδέσεις TCP
  - ServerSocketChannel για ακροατές εισερχόμενων TCP συνδέσεων (web server). Κάθε σύνδεση δημιουργεί ένα SocketChannel



java.nio.Buffer

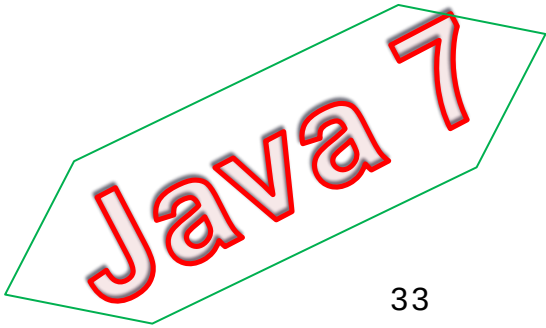


- `Abstract class`
  - Αναλαμβάνει την επικοινωνία με ένα channel
- Implementations
  - ByteBuffer, MappedByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer
- Methods
  - flip(): switches a Buffer from writing mode to reading mode.
  - rewind() sets the position back to 0, so you can reread all the data in the buffer
  - clear() the position is set back to 0 and the limit to capacity
  - compact() copies all unread data to the beginning of the Buffer
  - Buffer.mark() marks a given position in a Buffer.
  - You can reset to that position by calling the Buffer.reset()



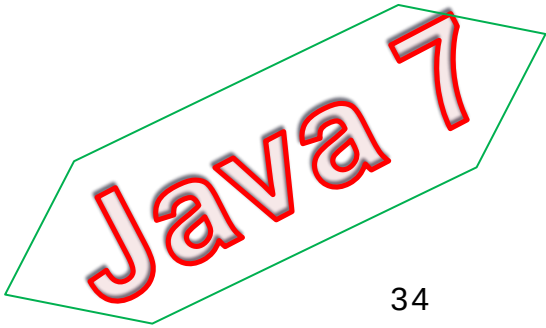
Παράδειγμα - Read

```
RandomAccessFile fin = new RandomAccessFile( "readbytes.txt“, “rw” );
FileChannel fc = fin.getChannel();
ByteBuffer buffer = ByteBuffer.allocate( 1024 );
int bytesRead = inChannel.read(buf); //write from channel to buffer
while (bytesRead != -1) {
    System.out.println("Read " + bytesRead);
    buf.flip(); //prepare buffer for read
    while(buf.hasRemaining()){
        System.out.print((char) buf.get());
    }
    buf.clear();
    bytesRead = inChannel.read(buf);
}
fin.close();
```



Παράδειγμα - Write

```
//WRITE
FileOutputStream fout = new FileOutputStream("writebytes.txt" );
FileChannel fc = fout.getChannel();
ByteBuffer buffer = ByteBuffer.allocate( 1024 );
for (int i=0; i<message.length; i++) {
    buffer.put( message[i] );
}
buffer.flip();  //prepare buffer for read
fc.write( buffer ); //read from buffer to channel
```



Διαχείριση Εξαιρέσεων

Διαχείριση λαθών – Εξαιρέσεις

- Δημιουργία
- Ανίχνευση
- Διαχείριση
- Βασικοί τύποι εξαιρέσεων
- Δημιουργία τύπων εξαίρεσης

Αντιμετώπιση λαθών

- Υπάρχουν λάθη χρόνου εκτέλεσης (run time errors) τα οποία δεν ανιχνεύονται στη μεταγλώττιση.
  - Αναζήτηση στοιχείου έξω από τα όρια ενός πίνακα
  - Άνοιγμα αρχείου που δεν υπάρχει
  - Κλήση αναφοράς σε null
- Θα πρέπει να αντιμετωπίζονται όταν το πρόγραμμα εκτελείται
- Ο κώδικας που δημιουργεί κάποιο λάθος μεταδίδει πληροφορία στον κώδικα που καλείται να το αντιμετωπίσει. Αυτό γίνεται με τις ***Εξαιρέσεις – Exceptions***
- Σηματοδοτούν εμφάνιση συνθηκών ιδιαίτερης μεταχείρισης και διακόπτουν τη συνήθη ροή του κώδικα

Παραδείγματα

- Αναζήτηση στοιχείου έξω από τα όρια ενός πίνακα

```
ArrayList<String> messages = new ArrayList<String>();
messages.add(“Hello”);
messages.add(“How are you?”);
System.out.println(messages.get(0));
System.out.println(messages.get(2)); //ArrayIndexOutOfBoundsException
```

- Αναφορά σε null αντικείμενο

```
Human[] anthrwpoi = new Human[5];
anthrwpoi[0] = new Human(“Bill”,”Gates”,50);
anthrwpoi[2].setName(“George”); //NullPointerException
```

- Άνοιγμα αρχείου που δεν υπάρχει

```
FileReader fro = new FileReader( "myFile.txt" ); //FileNotFoundException
```

Οι εξαιρέσεις είναι αντικείμενα

- Αντί να ελέγχουμε στον κώδικα για κάποιο συγκεκριμένο λάθος σε όσα σημεία μπορεί να εμφανιστεί, μπορούμε να προσθέσουμε ***ένα μόνο*** μπλόκ κώδικα που αποκαλείται διαχειριστής εξαίρεσης (Exception Handler)
- Αν συμβεί εξαίρεση, η ροή του προγράμματος στο μπλοκ διακόπτεται και ο έλεγχος περνά στο διαχειριστή εξαίρεσης
- Αυτός ανιχνεύει τον τύπο του αντικειμένου εξαίρεσης
- Αν κάποια εξαίρεση δεν τη διαχειριστούμε παίρνουμε μήνυμα από το μεταγλωττιστή

Παράδειγμα

- Θέλουμε να δημιουργήσουμε μια μέθοδο που να αθροίζει όσους αριθμούς δώσει ο χρήστης. Να συνεχίζει μέχρι ο χρήστης να δώσει το σύμβολο ‘=‘.

```
public static void main(String[] args) {
        Main m=new Main();
        double z=m.sumNumbers();
        System.out.println("Sum is:"+z);
    }
public double sumNumbers(){
    double sum=0;
    while(????)
                sum+=readDouble();
    return sum;
}
public double readDouble(){
    Scanner in =new Scanner(System.in);
    return in.nextDouble();
}
```

- Τι θα βάλω στη συνθήκη του while; Τι θα γίνει αν ο χρήστης δώσει ‘=‘; Τι θα γίνει αν δώσει κάτι που δεν είναι αριθμός;

Αναλυτικά με τις εξαιρέσεις

- Όταν εμφανίζεται μία εξαίρεση σταματά η εκτέλεση της μεθόδου
- Δεν υπάρχει η απαραίτητη πληροφορία στο scope της μεθόδου
- Ανεβαίνουμε και αντιμετωπίζουμε το πρόβλημα σε κάποιο «υψηλότερο» scope. «***Ρίχνουμε***» μια εξαίρεση (throw)
  - Ένα αντικείμενο εξαίρεσης (Exception object) δημιουργείται στον σωρο (heap) με χρήση του τελεστή new όπως κάθε άλλο Java object
  - Διακόπτεται η εκτέλεση της μεθόδου και κρατιέται μια αναφορά στο Exception object σε κάποια περιοχή της μνήμης
  - Αναλαμβάνει ο μηχανισμός διαχείρισης της εξαίρεσης (Exception handling mechanism) που πρέπει να βρει το κατάλληλο μέρος για να συνεχίσει να εκτελεί κώδικα

Αντικείμενα εξαιρέσεων

- Το keyword throw προκαλεί τα ακόλουθα γεγονότα:
  - εκτελεί την έκφραση new και δημιουργεί ένα αντικείμενο που δεν θα υπήρχε κανονικά
  - Το αντικείμενο αυτό επιστρέφεται στην ουσία από την μέθοδο ή το block μέσα στο οποίο δημιουργήθηκε μολονότι η μέθοδος δεν έχει δηλωθεί να επιστρέφει παραμέτρους αυτού του τύπου και τα blocks δεν έχουν φυσικά παραμέτρους που να επιστρέφουν
  - Το πρόγραμμα βγαίνει από τη μέθοδο ή το block.
- Τα αντικείμενα exception που «ρίχνονται» μπορεί να είναι ***διάφορων*** τύπων, ανάλογα το είδος της εξαίρεσης
- Κάθε αντικείμενο κωδικοποιεί στα πεδία του όλη την πληροφορία που αφορά την εξαίρεση, ώστε ο μηχανισμός διαχείρισης εξαιρέσεων στα ψηλότερα επίπεδα να μπορεί να πάρει τις κατάλληλες αποφάσεις

Προδιαγραφές εξαιρέσεων

- Αν κάποιος πρόκειται να χρησιμοποιήσει τον κώδικά μας τότε θα πρέπει να ξέρει τις εξαιρέσεις που μπορεί να παράγονται
- Σε κάθε μέθοδο που ρίχνει εξαιρέσεις χρησιμοποιούμε τη δήλωση ***throws***

```
void f() throws tooBig, tooSmall, divZero {
   // κώδικας της μεθόδου
}
```

- Αν κάποιος καλέσει στον κώδικά του τη μέθοδό μας θα πρέπει να τη βάλει μέσα σε ***try*** block.
- Σε αντίθετη περίπτωση ο μεταγλωττιστής εκδίδει μηνύματα λάθους. Αυτό εξασφαλίζει διαχείριση εξαιρέσεων κατά τη μεταγλώττιση.
- Τέτοιες προδιαγραφές υπάρχουν ήδη σε πολλές μεθόδους κλάσεων της Java

Δημιουργία νέων τύπων εξαιρέσεων

- Κάθε τύπος εξαίρεσης που δημιουργούμε πρέπει να κληρονομεί από κάποιο υπάρχουσα τάξη.
- Συνήθως αυτή που είναι νοηματικά κοντά. Αλλιώς από την Exception

```
class MyException extends Exception {
    public myException() {
    // κενός κατασκευαστής
    }
    public myException(String ms) {
    // κατασκευαστής με όρισμα
    }
    // specific fields or methods may exist
}
```

Παράδειγμα – έλεγχος εισόδου

- Στη μέθοδο readDouble() ελέγχουμε τι δίνει ο χρήστης

```
public double readDouble() throws Exception{
      Scanner in =new Scanner(System.in);
       if(in.hasNextDouble()){
          return in.nextDouble();
       }
       else throw new Exception();
}
```

- Εναλλακτικά μπορούμε να περάσουμε ένα μήνυμα στην εξαίρεση

```
public double readDouble() throws Exception{
      Scanner in =new Scanner(System.in);
       if(in.hasNextDouble()){
          return in.nextDouble();
       }
       else throw new Exception("Not a number");
}
```

Παράδειγμα – έλεγχος εισόδου (2)

- Μπορούμε να χρησιμοποιήσουμε δική μας εξαίρεση;
- Στο αρχείο MySpecialException.java

```
public class MySpecialException extends Exception{

}
```

- Οπότε

```
public double readDouble() throws MySpecialException{
      Scanner in =new Scanner(System.in);
       if(in.hasNextDouble()){
           return in.nextDouble();
       }
       else throw new MySpecialException();
}
```

Σύλληψη εξαίρεσης

- Όταν ρίχνουμε μία εξαίρεση πρέπει στα υψηλότερα επίπεδα κάποιο μπλοκ κώδικα να την ανιχνεύσει και να την διαχειριστεί.
- ***Προστατευόμενη περιοχή*** είναι ένα κομμάτι κώδικα που μπορεί να ρίξει εξαιρέσεις και που ακολουθείται από κώδικα που διαχειρίζεται αυτές τις εξαιρέσεις.
- Μια προστατευόμενη περιοχή μπορεί να ανιχνεύσει ένα ή περισσότερους τύπους εξαιρέσεων και
  - να τους διαχειριστεί ή
  - να τους ρίξει ακόμη πιο πάνω (με νέο throw)

Παράδειγμα

- Μη διαχείριση σφάλματος εισόδου

```
public double sumNumbers() throws MySpecialException{
      double sum=0;
      sum+=readDouble();
      return sum;
}
 public static void main(String[] args) throws MySpecialException{
        Main m=new Main();
        double z=m.sumNumbers();
        System.out.println("Sum is:"+z);
    }
```

- Έτσι αποφεύγουμε να χειριστούμε την εξαίρεση

Προστατευόμενη περιοχή - *try*

- Ένα τέτοιο μπλοκ ονομάζεται try block αφού εκεί δοκιμάζουμε να κάνουμε κλήσεις σε διάφορες «επικίνδυνες μεθόδους»
- Το try block είναι ένα κανονικό scoper που περικλείεται από την κωδική λέξη try

```
try {
   // κώδικας που παράγει εξαιρέσεις
}
```

- Προστασία από εξαιρέσεις: Βάζουμε όλον τον «επικίνδυνο» κώδικα σε ένα try block και πιάνουμε όλες τις εξαιρέσεις στο ίδιο μέρος.

Διαχειριστές εξαιρέσεων - ***catch***

- Κάθε εξαίρεση που ανιχνεύεται μέσα στο try καταλήγει στον αντίστοιχο κώδικα (***catch*** block) που θα την εξυπηρετήσει. Τα catch ελέγχονται διαδοχικά.
- Παράδειγμα:

```
try {
   // κώδικας που παράγει εξαιρέσεις
} catch (type1 id1) {
   // χειρισμός εξαιρέσεων τύπου 1
} catch (type2 id2) {
   // χειρισμός εξαιρέσεων τύπου 2
} catch (type3 id3) {
   // χειρισμός εξαιρέσεων τύπου 3
}
```

- Αν πολλές μέθοδοι στο try ρίχνουν τον ίδιο τύπο εξαίρεσης, τότε χρειαζόμαστε μόνο έναν exception handler για αυτόν τον τύπο.

Η τάξη Exception

- Ένας διαχειριστής εξαιρέσεων με όρισμα τύπου Exception συλλαμβάνει όλες τις εξαιρέσεις.

```
catch(Exception e) {
        System.out.println("caught an exception");
}
```

- Γι' αυτό μπαίνει στο τελευταίο catch στη σειρά για να πιάσει κάθε άλλη εξαίρεση.
- Η Exception και η Throwable προσφέρουν τις ακόλουθες μεθόδους:
  - String getMessage()
  - String toString() //τυπώνει το όνομα της εξαίρεσης
  - void printStackTrace() //τυπώνουν το σωρό
  - void printStackTrace(PrintStream) //μεθόδων που κλήθηκαν

Παράδειγμα

- Διαχείριση σφάλματος εισόδου

```
public double sumNumbers(){
      double sum=0;
       try{
                sum+=readDouble();
       }
       catch(MySpecialException ex){              //θα συμβεί αν ο χρήστης δεν δώσει αριθμό
            System.err.println(ex.toString());    //τυπώνει το όνομα της κλάσης εξαίρεσης
       }
       catch(Exception ex){                       //θα συμβεί σε οποιαδήποτε άλλη περίπτωση λάθους
           System.err.println(ex.getMessage());   //τυπώνει το μήνυμα της εξαίρεσης
       }
      return sum;
}
```

- Η MySpecialException είναι ειδικότερη από την Exception

Το block ***finally***

- Όταν υπάρχει κάποιο κομμάτι κώδικα που θα πρέπει να εκτελεστεί είτε συμβαίνει ένα Exception σε ένα try μπλοκ είτε όχι βάζουμε τον κώδικα σε ένα finally μπλοκ ώς εξής:

```
static Switch sw = new Switch();
try {
    sw.on();
    …//επικίνδυνος κώδικας
} catch (A a) {
    ...
} catch (B b) {
    ...
} finally {
    sw.off(); //κώδικας που τρέχει πάντοτε και κάνει reset
}
```

Τερματισμός ή ανάνηψη (συνέχιση)

- Τερματισμός: Θεωρούμε ότι τα λάθη είναι κρίσιμα και δε συνεχίζουμε την εκτέλεση του κώδικα από το σημείο που ρίχθηκε η εξαίρεση
- Ανάνηψη: Προσπαθούμε να συνεχίσουμε την εκτέλεση του προγράμματος από εκεί που διακόπηκε

```
while (true) {
   try {
            //κώδικας που παράγει εξαιρέσεις
   }
   catch (Type1 id1) { //διορθώνουμε την εξαίρεση
   continue;}
   ...
   catch (TypeN idN) { // διορθώνουμε την εξαίρεση
   continue;}
   // κώδικας που εκτελείται αν δεν υπάρχει εξαίρεση
   break;
}
```

Παράδειγμα

- Ας διαχωρίσουμε τις εξαιρέσεις

```
public double readDouble() throws MySpecialException, Exception {
        Scanner in = new Scanner(System.in);
        if (in.hasNextDouble()) {
            return in.nextDouble();
        } else if (in.hasNext() && in.next().equals("=")){
            throw new MySpecialException();    // αν ο χρήστης δώσει ‘=‘
        }
        else{
            throw new Exception("Not a number");     // αν ο χρήστης δώσει
    οτιδήποτε άλλο
        }
    }
```

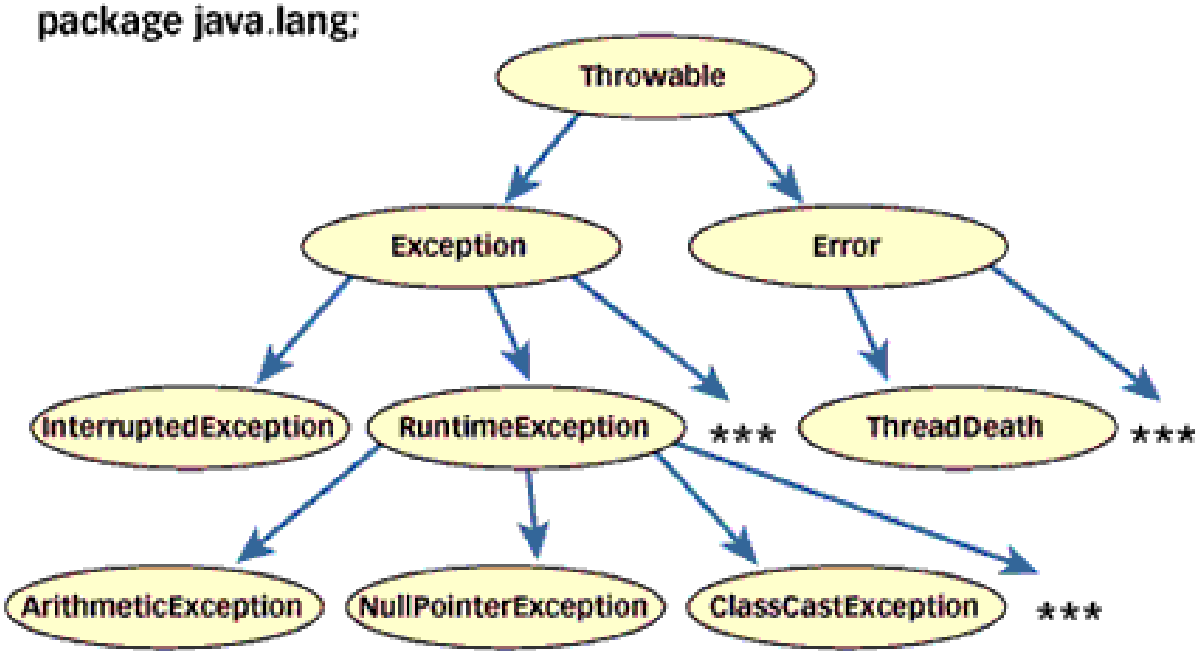
Παράδειγμα (2)

- Τερματισμός επανάληψης

```
public double sumNumbers() {
    double sum = 0;
    while (true) {
        try {
            sum += readDouble();
        } catch (MySpecialException ex) {
            break;
        } catch (Exception ex) {
            System.err.println(ex.getMessage());
        }
    }
    return sum;
}
```

Ιεραρχία εξαιρέσεων

- Η Java περιέχει την τάξη ***Throwable*** που περιγράφει οτιδήποτε μπορεί να ριχθεί σαν Exception
- Άλλες εξαιρέσεις ορίζονται στα πακέτα util, net και io.



Παράδειγμα

```
public class ExceptionMethods {
    public static void main(String args[]) {
        try {
            throw new Exception("Here's my Exception");
        } catch (Exception e) {
            System.out.println("Caught Exception");
            System.out.println( e.getMessage());
            System.out.println( e.toString());
            System.out.println( e.printStackTrace());
        }
    }
}
```

Η δύο τελευταίες κλήσεις τυπώνουν

java.lang.Exception: Here's my Exception

java.lang.Exception: Here's my Exception
at ExceptionMethods.main

Κληρονομικότητα και εξαιρέσεις

- Όταν σε μια απόγονη τάξη κάνουμε override μια μέθοδο της γονικής τάξης, τότε η μέθοδος δεν μπορεί να ρίχνει επιπλέον εξαιρέσεις.

```
class Test {
  String method (String s){
     if (s==null)
        throw new NullPointerException();
     else
        return s;}}

class SubTest extends Test{
String method (String s) throws Exception{
     if (s==null)
        throw new Exception();
     else
        return s;}}
```

overridden method does not throw java.lang.Exception

Παράδειγμα 1

- Ανάγνωση από πληκτρολόγιο
  - Δημιουργούμε ένα αντικείμενο με χρήση του ρεύματος εισόδου in (τάξη InputStream).

    BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
  - Και καλούμε τη μέθοδο ανάγνωσης γραμμής

```
try{
        line=stdin.readLine();
}
catch (IOException e){
        line=“”; //σε περίπτωση αποτυχίας γίνεται κενό
}
```

Παράδειγμα 2

- Μετατροπή String σε int

```
try {
   x= Integer.parseInt(line);
}
catch(NumberFormatException e) {
   x=0; //σε περίπτωση αποτυχίας γίνεται 0
}
```

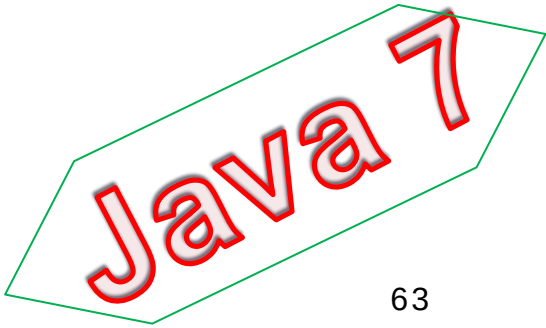
Java 7

Αυτόματη διαχείριση πόρων

```
public void oldTry() {
       try {
         fos = new FileOutputStream(“a.txt");
         dos = new DataOutputStream(fos);
         dos.writeUTF("Java 6");
       } catch (IOException e) {
             e.printStackTrace();
       } finally {
             try {
                 fos.close();
                 dos.close();
                 } catch (IOException e) {
                 // log the exception
             }
         }
       }
```

try-with-resources

```
public void newTry() {
   try (
      FileOutputStream fos = new
                FileOutputStream(“a.txt");
      DataOutputStream dos = new
                DataOutputStream(fos)
      ) {
      dos.writeUTF("Java 7");
   }
   catch (IOException e) {
             // log the exception
   }
}
```



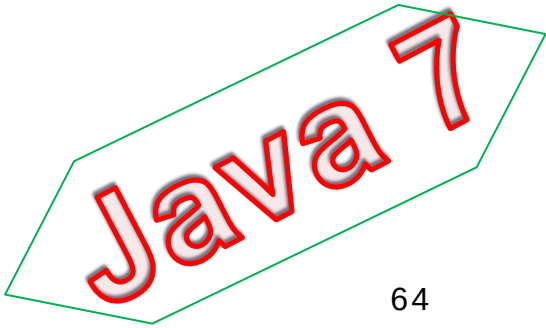
Ο τελεστής <>

- diamond operator
- Αντί για

```
HashMap<String, Stack<String>> map =
    new HashMap<String, Stack<String>>();
```

- Μπορώ να γράψω:

```
HashMap<String, Stack<String>> map =
    new HashMap<>;
```



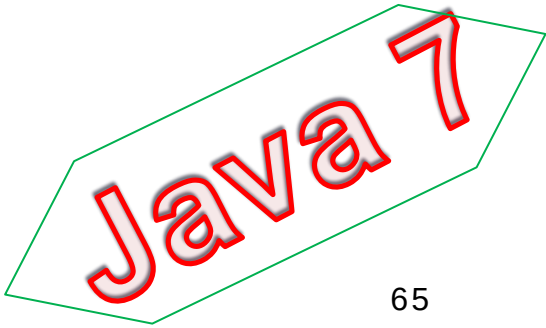
Strings στη `switch`

- Αντί για

```
if (input.equals(“yes”)) {
    return true;
} else if (input.equals(“no”)) {
    return false;
} else {
    askAgain();
}
```

- Γράφουμε

```
switch(input) {
    case “yes”: return true;
    case “no”: return false;
    default: askAgain();
}
```



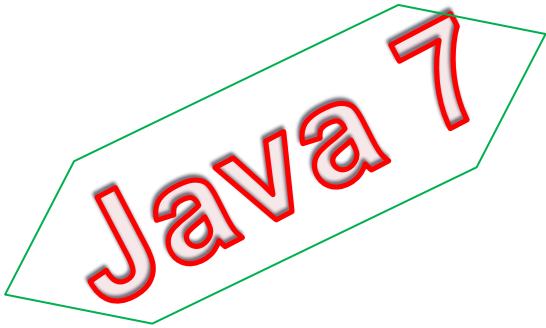
Διαχωριστικά χιλιάδων

- Αντί για

```
int million = 1000000;
```

γράφουμε

```
int million = 1_000_000;
```



Multi-catch

```
public void newMultiMultiCatch() {
    try {
        methodThatThrowsThreeExceptions();
    } catch (ExceptionOne e) {
        // deal with ExceptionOne
    } catch (ExceptionTwo | ExceptionThree e) {
        // deal with ExceptionTwo and ExceptionThree
    }
}
```