

## Rassegna API - 2: eccezioni e file

Stefano Mizzaro

Dipartimento di matematica e informatica  
Università di Udine  
<http://www.dimi.uniud.it/mizzaro/>  
mizzaro@uniud.it  
Programmazione, lezione 18,19  
5,11/5/2015

## Riassunto

- Programmazione strutturata
- OO
  - TDA, scambio messaggi, eredità, polimorfismo
- OO in Java
  - Classi astratte, interfacce, classi interne, package
- API
  - "Sguardo dall'alto", **Math**, **Object**, **System**, **String**

Stefano Mizzaro - API 2

2

## Oggi

- Eccezioni (cenni)
- API per i file, package `java.io`
  - Gestione file come oggetti atomici
  - Lettura e scrittura in un file
  - File di tipi primitivi
  - File ad accesso casuale

Stefano Mizzaro - API 2

3

## Eccezioni

- Quando si ha una situazione anomala in esecuzione...
  - `/0`, accesso ad array con indice errato (`ArrayIndexOutOfBoundsException`), lettura oltre fine file, ecc. ecc.
- ...l'interprete Java:
  - sospende l'esecuzione normale
  - getta un'eccezione: crea un'istanza di (una sottoclasse di) `java.lang.Exception`
  - (eventualmente) esegue codice per la gestione dell'eccezione

Stefano Mizzaro - API 2

4

## Gettare, catturare, rimbalzare

- Un'eccezione viene **gettata**
  - Durante l'esecuzione di un certo metodo, che avrà il suo record di attivazione sullo stack...
- Se viene **catturata**: viene gestita
- Altrimenti viene **rimbalzata**
  - al blocco più esterno (e al più esterno e...)
  - al chiamante (e al chiamante del chiamante...)
- Vediamole una alla volta...

Stefano Mizzaro - API 2

5

## Gettare

- Gettare = creare un'istanza di `java.lang.Exception` (o sottoclasse)
- Chi lo fa:
  - L'interprete Java
    - al verificarsi di una situazione anomala
  - Esplicitamente il programmatore
    - `throw <oggettoEccezione>`
    - Es.: `throw new Exception();`

Stefano Mizzaro - API 2

6

## Catturare

- Costruito `try/catch/finally`
- Esegue `<I>`
- Se `<I>` getta eccezione `e`,
  - `<I>` si interrompe
  - viene eseguita la prima `<Ii>` t.c. e `instanceof <Ei>`
- `<id>` è come un parametro formale
- `<If>` sempre eseguita alla fine

```

try {
  <I>
} catch (<E1> <id>) {
  <I1>
} catch (<E2> <id>) {
  <I2>
} ...
} catch (<En> <id>) {
  <In>
} finally {
  <If>
}
    
```

Stefano Mizzaro - API 2 7

## Rimbalzare (throws)

- Un'eccezione non catturata viene rimbalzata
  - Al blocco esterno (ricorsivamente)
  - Al metodo chiamante (ricorsivamente)
- Se un metodo `m()` rimbalza eccezioni va dichiarato:
 

```
public void m() throws E1, E2 {
```

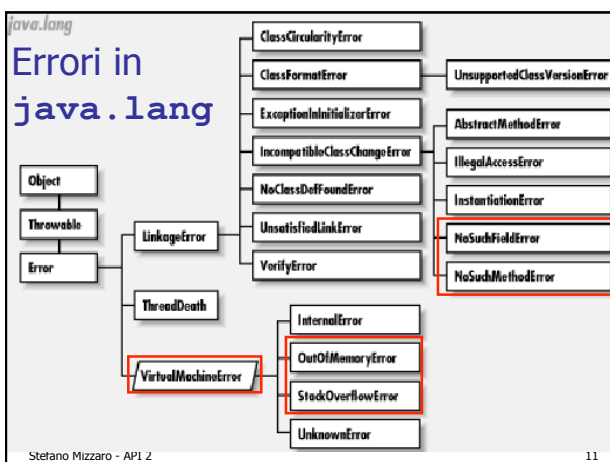
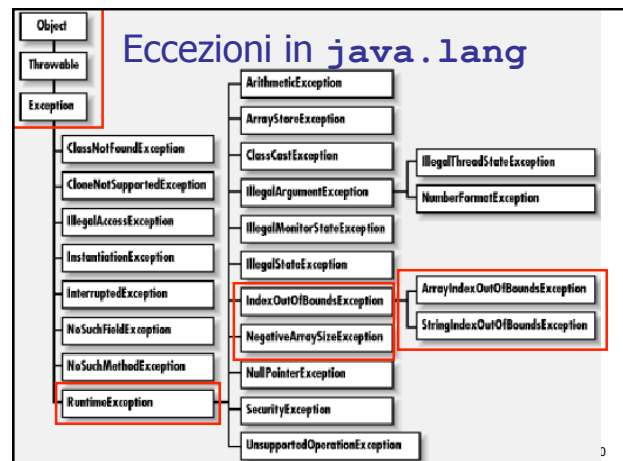
  - (se **checked**, ossia non `RuntimeException`)
  - (se **unchecked**, non serve – è come se il `throws` ci fosse comunque)
- Regola generale: Meglio catturare che rimbalzare

Stefano Mizzaro - API 2 8

## Eccezioni ed errori

- Eccezioni predefinite
- Throwable** ("oggetti gettabili")
  - Exception**
  - Error**: il programma termina
- Non catturate gli **Error**
- 100+ classi predefinite
- Vediamo la documentazione...

Stefano Mizzaro - API 2 9



## Eccezioni definite da noi

```

class EccezioneMia extends Exception {}

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

Stefano Mizzaro - API 2 12

## Utilità delle eccezioni

- Separare codice "normale" da situazioni anomale
- Maggiore leggibilità
- Meglio di "valore funzione = -1: ⇒ errore!"
- Ne riparlerete...
- Usate nella gestione dei file
  - Quando un programma lavora su un file può succedere che il file sia scomparso...

Stefano Mizzaro - API 2

13

## Per inciso, parole riservate

```

abstract boolean break byte case catch
char class const continue default do
double else extends false final
finally float for goto if implements
import instanceof int interface long
native new null package private
protected public return short static
super switch synchronized this throw
throws transient true try void
volatile while
  
```

Stefano Mizzaro - API 2

14

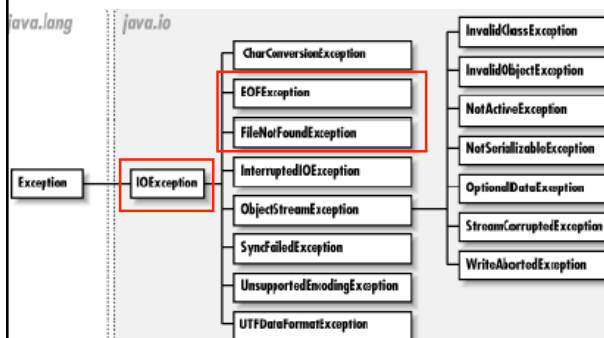
## Scaletta

- Eccezioni (cenni)
- I file, package `java.io`
  - Gestione file come oggetti atomici
  - Lettura e scrittura in un file
  - File di tipi primitivi
  - File ad accesso casuale

Stefano Mizzaro - API 2

15

## Eccezioni di `java.io`



Stefano Mizzaro - API 2

16

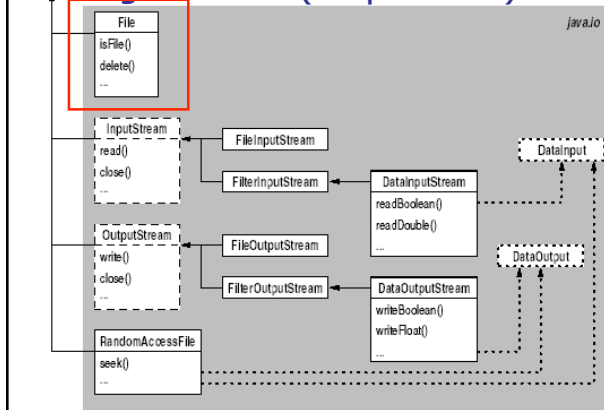
## Flussi e file

- Flusso (stream) = sequenza di dati
  - di input: da cui leggere
  - di output: su cui scrivere
- I file sono visti come flussi di dati
- Il package `java.io` definisce parecchie (>50) classi e interfacce per la gestione dei file (vediamo la documentazione)
- Vediamo le più semplici (e inefficienti...)
- Tramite esempi

Stefano Mizzaro - API 2

17

## `java.io` (un pezzetto)



## java.io.File (1/2)

- Un'istanza di `File` rappresenta un file/pathname all'interno di un programma
- I metodi (d'istanza) di `File` vedono un file come oggetto atomico (non vedono il contenuto)
- Associare l'istanza di `File` a un pathname (con il costruttore):
  - `public File(String)`
- Creare un `File` vuoto:
  - `public boolean createNewFile()`
  - (il "costruttore" non è un "creatore"; con solo il costruttore, il file fisico sul file system NON viene creato!!)

Stefano Mizzaro - API 2

19

## java.io.File (2/2)

- Verificare se un file esiste
  - `public boolean exists()`
- Cancellare un file
  - `public boolean delete()`
- Verificare se l'istanza rappresenta un file o una directory:
  - `public boolean isFile()`
  - `public boolean isDirectory()`
- Cfr. documentazione API

Stefano Mizzaro - API 2

20

## Esempio

```
import java.io.*;
class ProvaFile {
    public static void main (String[] args)
        throws IOException {
        File f = new File("pippo.txt");
        System.out.println(f.exists());
        f.createNewFile();
        System.out.println(f.exists());
        System.out.println(f.isFile());
        System.out.println(f.isDirectory());
        System.in.read();
        f.delete();
    }
}
```

```
>java ProvaFile
false
true
true
true
false
```

Stefano Mizzaro - API 2

21

## Scaletta

- Eccezioni (cenni)
- I file, package `java.io`
  - Gestione file come oggetti atomici
  - Lettura e scrittura in un file
  - File di tipi primitivi
  - File ad accesso casuale

Stefano Mizzaro - API 2

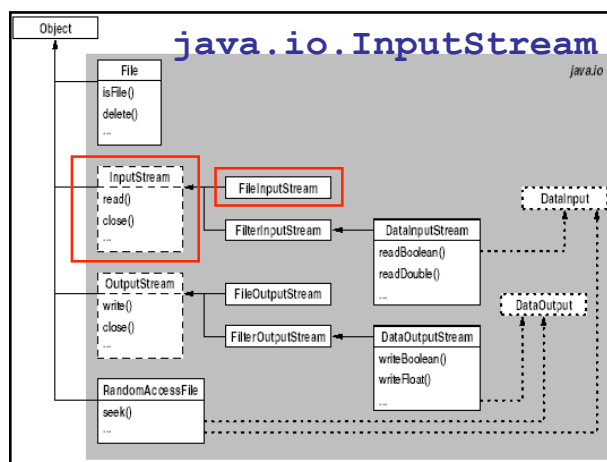
22

## Come si lavora sul contenuto di un file

- 3 passi
  - Apertura
  - Operazioni lettura/scrittura
  - Chiusura
- Gestione diversa per lettura e scrittura
- Vediamo prima la lettura

Stefano Mizzaro - API 2

23



## InputStream

E nei file ci sono caratteri...

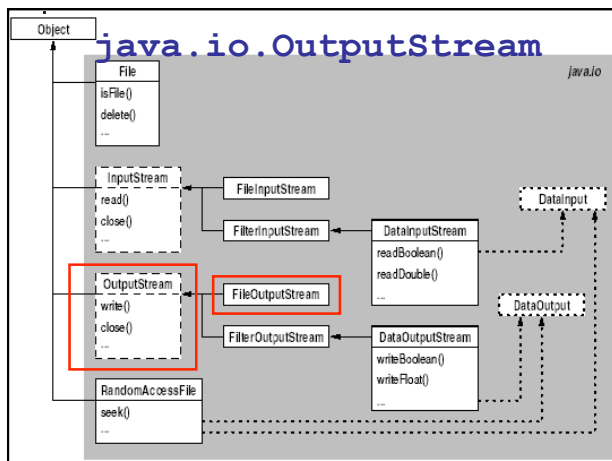
- Classe astratta
  - `public abstract int read():`
    - legge un byte dal flusso, restituisce il codice ASCII
    - Sovraccarico (`public int read(byte b[])`) e può leggere anche un array di byte
    - Restituisce -1 a fine file
  - `public void close():` chiude il flusso
  - `public long skip(long n):` salta il numero di byte specificato dal parametro

Stefano Mizzaro - API 2 25

## FileInputStream

- Sottoclasse di `InputStream`
- È quella da usare, i suoi oggetti sono flussi di input. Non definisce altri metodi; ovviamente:
  - implementa `read`
  - e definisce il costruttore, da usare per l'apertura (lo vediamo subito)
- Quindi si può leggere il contenuto di un file, carattere per carattere, uno dopo l'altro sequenzialmente

Stefano Mizzaro - API 2 26



## OutputStream

E nei file ci sono caratteri...

- Simmetria con `InputStream`
- Classe astratta
  - `public abstract void write(int b):`
    - scrive un byte sul flusso (8 bit meno significativi)
    - sovraccarico (`public void write(byte b[])`) e può scrivere anche un array di byte alla volta
  - `public void close():` chiude il flusso. È molto importante chiudere i flussi di output per non perdere i dati non ancora salvati

Stefano Mizzaro - API 2 28

## FileOutputStream

- Sottoclasse di `OutputStream`
- È quella da usare, i suoi oggetti sono flussi di output. Non definisce altri metodi; ovviamente:
  - implementa `write`
  - e definisce il costruttore, da usare per l'apertura (lo vediamo subito)
- Quindi si può scrivere il contenuto di un file, carattere per carattere, uno dopo l'altro sequenzialmente

Stefano Mizzaro - API 2 29

## Apertura file

- I costruttori di `FileInputStream` e `FileOutputStream` aprono un file (lo "preparano") in lettura e scrittura
- Parametro:
  - un oggetto della classe `File`:
    - `FileInputStream f;`  
`f = new FileInputStream(new File("pippo/pluto.txt"));`
  - oppure una stringa (pathname del file):
    - `FileInputStream f;`  
`f=new FileInputStream("pippo/pluto.txt");`

Stefano Mizzaro - API 2 30

## Esempio 1: VediFile.java

- Scriviamo un programma per visualizzare il contenuto di un file di testo
- Uso:

```
>java VediFile pippo.txt
```

Stefano Mizzaro - API 2

31

## VediFile.java (1/2)

```
import java.io.*;
public class VediFile {
    public static void main(String args[]) {
        FileInputStream f;
        int c;
        if (args.length != 1)
            System.out.println(
                "Uso: java VediFile <file>");
        else {
            try {
                f = new FileInputStream(args[0]);
            } catch (FileNotFoundException e) {
                System.out.println(
                    "Errore in apertura file");
                return;
            }
        }
    }
}
```

Stefano Mizzaro - API 2

32

## VediFile.java (2/2)

```
        try {
            c = f.read();
            while (c > 0) {
                System.out.print((char) c);
                c = f.read();
            }
        } catch (IOException e) {
            System.out.println(
                "Errore in lettura file");
        }
        try {
            f.close();
        } catch (IOException e) {
            System.out.println(
                "Errore in chiusura file");
        }
    }
}
```

Stefano Mizzaro - API 2

33

## Commenti

- Leggiamo carattere per carattere dal file e scriviamo su standard output
- Cast perché `read` restituisce un `int`
  - Esercizio: cosa viene visualizzato se tolgo il cast?
- A fine file la `read` restituisce `-1`
- Le eccezioni vanno gestite/catturate
- `try/catch` per gestire tutte le possibili eccezioni

Stefano Mizzaro - API 2

34

## Es. 2 : CopiaFile.java

- Programma che copia un file in un altro
- Legge, carattere per carattere, da un `(File) InputStream` e...
- ... scrive, carattere per carattere, su un `(File) OutputStream`
- Uso:

```
>java CopiaFile pippo.txt pluto.txt
```

Stefano Mizzaro - API 2

35

## CopiaFile.java

```
import java.io.*;
public class CopiaFile {
    public static void main(String args[])
        throws FileNotFoundException, IOException {
        FileInputStream in;
        FileOutputStream out;
        int c;
        if (args.length != 2)
            System.out.println(
                "Uso: java CopiaFile <fileIn> <fileOut>");
        else {
            in = new FileInputStream(args[0]);
            out = new FileOutputStream(args[1]);
            c = in.read();
            while (c > 0) {
                out.write((char) c);
                c = in.read();
            }
            in.close();
            out.close();
        }
    }
}
```

## Commenti

- Non abbiamo catturato le eccezioni (male! ⇒Ex.)
- Il file di output viene creato dal costruttore di **FileOutputStream**
  - Non è come per **File**
  - Ex.: verificare che è proprio così
- Il corpo del ciclo ha la stessa struttura di **VediFile**

Stefano Mizzaro - API 2

37

## Scaletta

- Eccezioni (cenni)
- I file, package **java.io**
  - Gestione file come oggetti atomici
  - Lettura e scrittura in un file
  - File di tipi primitivi
  - File ad accesso casuale

Stefano Mizzaro - API 2

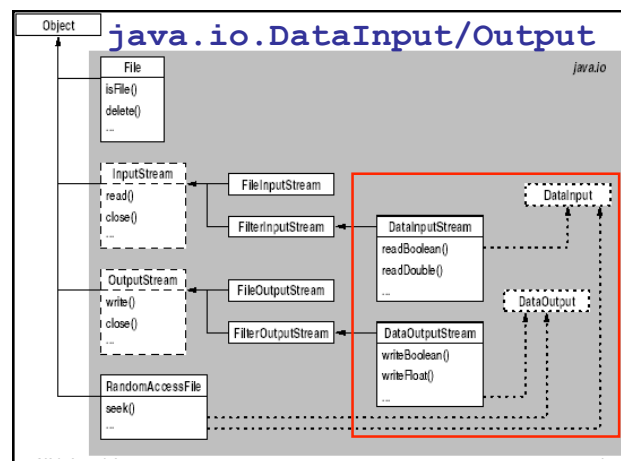
38

## I file e i tipi primitivi

- Finora: lettura/scrittura byte per byte, carattere per carattere
- Spesso è comodo usare i tipi primitivi (leggere e scrivere **double**, **int**, **boolean**, ...).
- Si usano:
  - **DataInputStream** (che implementa **DataInput**)
  - **DataOutputStream** (che implementa **DataOutput**)

Stefano Mizzaro - API 2

39



## DataInput (Stream) e DataOutput (Stream)

- Le interfacce **DataInput** e **DataOutput** contengono metodi del tipo **readBoolean()**, **readByte()**, **writeDouble()**, **writeShort()**, ...
- **DataInputStream** e **DataOutputStream**
  - simili alle classi viste finora
  - però i costruttori non sono sovraccarichi e vogliono come parametro un oggetto di tipo, rispettivamente, **InputStream** e **OutputStream** (non una stringa)

Stefano Mizzaro - API 2

41

## Esempio: I/O di tipi primitivi

```
import java.io.*;
public class DataFile {
    public static void main (String[] args)
        throws IOException {
        DataOutputStream out;
        DataInputStream in;
        out = new DataOutputStream(new
            FileOutputStream("tmp"));
        out.writeBoolean(false);
        out.writeDouble(12.34);
        out.writeChar('a');
        out.close();
        in = new DataInputStream(new
            FileInputStream("tmp"));
        System.out.println(in.readBoolean());
        System.out.println(in.readDouble());
        System.out.println(in.readChar());
        in.close();
    }
}
```

Stefano Mizzaro - API 2

42

## Commenti

- Proviamo a eseguirlo
- Proviamo a visualizzare il file creato...
- Notate che non "parliamo alla classe (interfaccia) base"
  - `DataOutputStream` non `DataOutput`
  - `DataInputStream` non `DataInput`
- ... perché le due interfacce non hanno il metodo `close()`
  - Avremmo errore in compilazione -> provare (Ex.)
- No eccezioni... -> Ex.

Stefano Mizzaro - API 2

43

## Scaletta

- Eccezioni (cenni)
- I file, package `java.io`
  - Gestione file come oggetti atomici
  - Lettura e scrittura in un file
  - File di tipi primitivi
  - File ad accesso casuale

Stefano Mizzaro - API 2

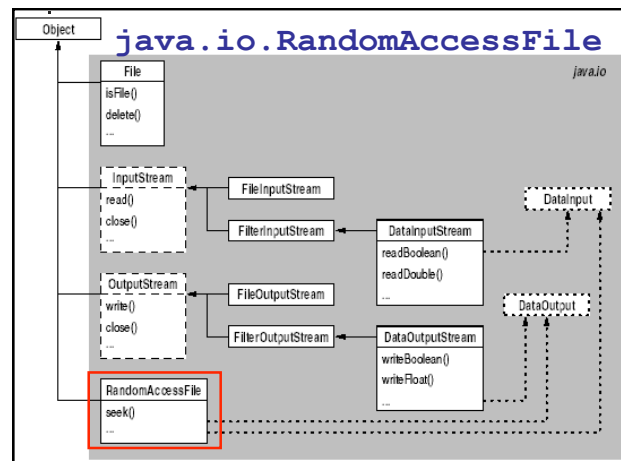
44

## I file ad accesso casuale (diretto)

- Finora solo accesso sequenziale
  - Impossibile "tornare indietro" (solo chiudendo e riaprendo il flusso)
- Accesso diretto, non sequenziale
  - "come un array" (però su disco)
- `java.io.RandomAccessFile`
- implementa `DataInput` e `DataOutput`
- ha un metodo `seek(int)` per posizionarsi all'n-esimo byte del file (1° è zero)
- Ha anche altri metodi (tutti quelli di `DataInput` e `DataOutput`)

Stefano Mizzaro - API 2

45



## Esempio: lettura da file ad accesso casuale

- Scriviamo 6 caratteri su un file
- Poi li leggiamo usando l'accesso casuale
  - in ordine inverso
  - dal quinto al primo

Stefano Mizzaro - API 2

47

## FileAccessoCasuale.java

```
import java.io.*;
class FileAccessoCasuale {
    public static void main (String[] args)
        throws IOException, EOFException {
        FileOutputStream out = new
            FileOutputStream("tmp");
        for (char c = 'a'; c <= 'f'; c++)
            out.write(c);
        out.close();
        RandomAccessFile f = new
            RandomAccessFile("tmp", "r");
        for (int i = 4; i >= 0; i--) {
            f.seek(i);
            System.out.println((char) f.readByte());
        }
    }
}
```

Stefano Mizzaro - API 2

48



## Commenti

- Eccezioni...
- Costruttore con 2 parametri:
  - "r" indica lettura (read)
  - "rw" indicherebbe lettura & scrittura (read & write)
- Evitare operazioni continue in memoria secondaria (es.: ordinamento di file)...
- ... ma evitare anche di caricare file enormi in memoria centrale...

Stefano Mizzaro - API 2

49

## Riassunto

- Eccezioni
  - Gettare, catturare, rimbalzare
  - try/catch/finally
  - throw, throws
- Flussi e file
  - File, InputStream, FileInputStream, OutputStream, FileOutputStream, DataInputStream, DataOutputStream, RandomAccessFile
  - Ci sarebbe molto altro... (Reader, Writer, BufferedReader/Writer, ...), nio, ... --> "Arrangiatevi!" (doc. API)
- Prox. lezione: API x GUI (AWT)

Stefano Mizzaro - API 2

50