

Rassegna API - 2: eccezioni e file

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
<http://www.dimi.uniud.it/~mizzaro>
mizzaro@dimi.uniud.it
Programmazione, lezione 21
28 aprile 2004

Riassunto

- Programmazione strutturata
- TDA
- OO
 - Scambio messaggi, eredità, polimorfismo
- OO in Java
 - Classi astratte, interfacce, classi interne, package
- API
 - **Math, Object, System, String**

Stefano Mizzaro - API 2

2

Scaletta

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

Stefano Mizzaro - API 2

3

Eccezioni

- Quando si ha una situazione anomala in esecuzione (/0, lettura oltre fine file, ...):
 - si sospende l'esecuzione normale
 - si getta un'eccezione: crea un'istanza di (una sottoclasse di) **java.lang.Exception**
 - si esegue codice per la gestione dell'eccezione

Stefano Mizzaro - API 2

4

Gettare, catturare, rimbalzare

- Un'eccezione viene gettata
- Se viene catturata: viene gestita
- Altrimenti viene rimbalzata
 - al blocco più esterno (e al più esterno e...)
 - al chiamante (e al chiamante del chiamante...)

Stefano Mizzaro - API 2

5

Gettare

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

Stefano Mizzaro - API 2

6

Catturare

- 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 m() rimbalza eccezioni va dichiarato:


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

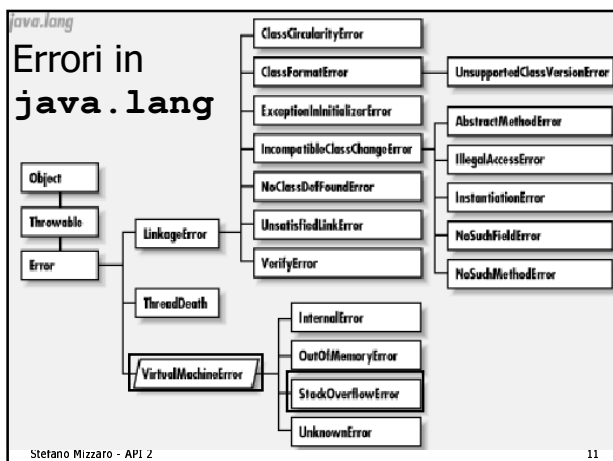
 - (se checked, ossia non RuntimeException)
- Meglio catturare...

Stefano Mizzaro - API 2 8

Eccezioni predefinite

- Throwable
 - 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

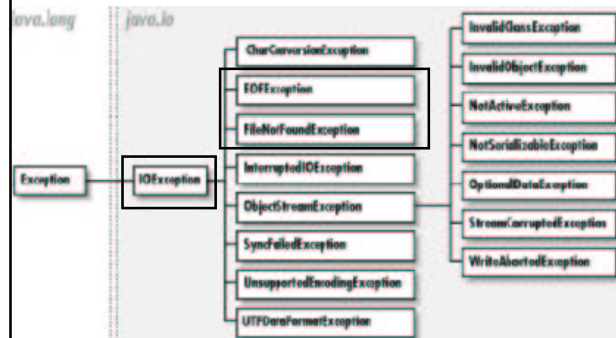
Utilità delle eccezioni

- Separare codice "normale" da situazioni anomale
- Maggiore leggibilità
- Meglio di "valore funzione = 1: ⇒ errore!"
- Ne riparlerete...

Stefano Mizzaro - API 2

13

Eccezioni di `java.io`



Stefano Mizzaro - API 2

14

Scaletta

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

Stefano Mizzaro - API 2

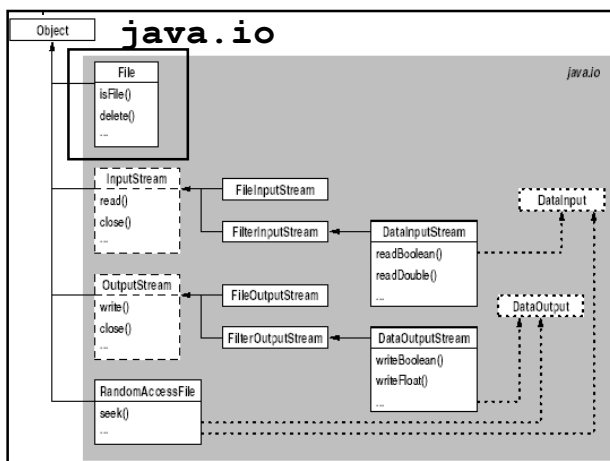
15

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...)

Stefano Mizzaro - API 2

16



`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)`
- Verificare se un file esiste
 - `public boolean exists()`

Stefano Mizzaro - API 2

18

java.io.File (2/2)

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

Stefano Mizzaro - API 2

19

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();
    }
}
```

Stefano Mizzaro - API 2

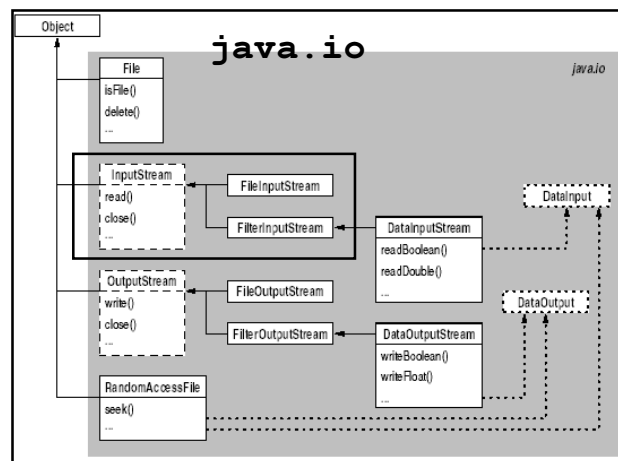
20

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

21



InputStream

- Classe astratta
 - `public abstract int read():`
 - legge un byte dal flusso, ritorna 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

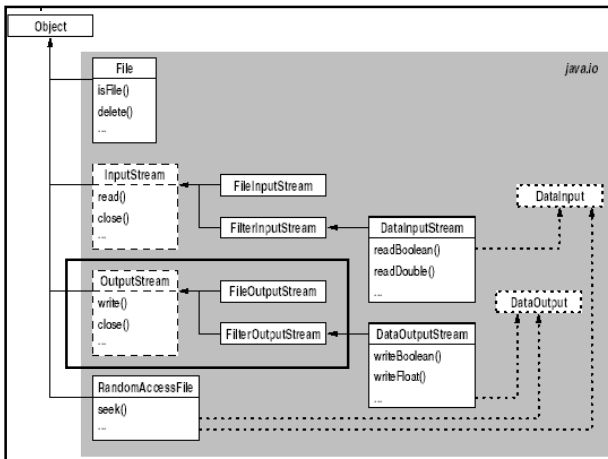
23

FileInputStream

- Sottoclasse di `InputStream`
- È quella da usare, i suoi oggetti sono flussi di input. Non definisce altri metodi
- (ovviamente implementa `read`)
- Quindi si può leggere il contenuto di un file, carattere per carattere

Stefano Mizzaro - API 2

24



OutputStream

- 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 importante chiudere i flussi di output per non perdere i dati non ancora salvati

Stefano Mizzaro - API 2

26

FileOutputStream

- Sottoclasse di **OutputStream**
- Non definisce altri metodi
- È quella da usare, i suoi oggetti sono flussi di output

Stefano Mizzaro - API 2

27

Apertura file

- I costruttori di **FileInputStream** e **FileOutputStream** aprono un file 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

28

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

29

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

30

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

31

Commenti

- Leggiamo carattere per carattere dal file e scriviamo su standard output
- Cast perché `read` restituisce un `int`
- Le eccezioni vanno catturate
- `try/catch` per gestire tutte le possibili eccezioni

Stefano Mizzaro - API 2

32

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

33

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!)
- Il file di output viene creato dal costruttore di `FileOutputStream`
- Il corpo del ciclo ha la stessa struttura di `VediFile`

Stefano Mizzaro - API 2

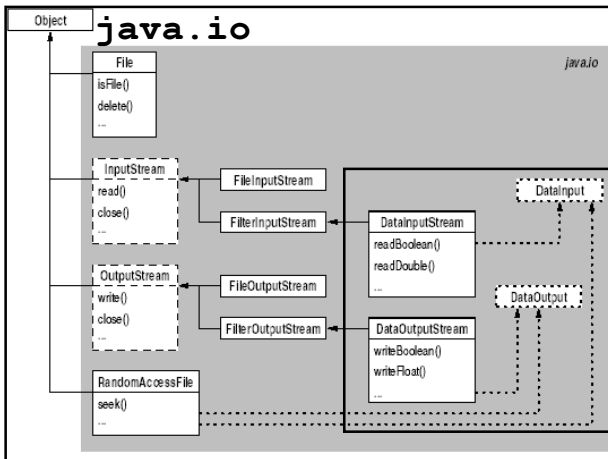
35

I file e i tipi primitivi

- Finora: lettura/scrittura byte per byte, carattere per carattere
- È comodo usare i tipi primitivi (leggere e scrivere `double`, `boolean`, ...):
 - `DataInputStream` (che implementa `DataInput`)
 - `DataOutputStream` (che implementa `DataOutput`)

Stefano Mizzaro - API 2

36



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

38

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('q');
        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();
    }
}
```

Commenti

- Esecuzione
- No eccezioni...

Stefano Mizzaro - API 2

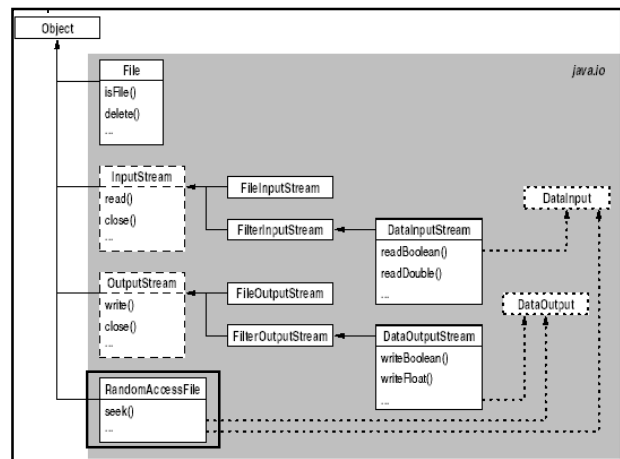
40

I file ad accesso casuale (diretto)

- Accesso diretto, non sequenziale
- 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

41



Esempio: lettura da file ad accesso casuale

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

Stefano Mizzaro - API 2

43

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

44

Commenti

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

Stefano Mizzaro - API 2

45

Riassunto

- Eccezioni
 - Gettare, catturare, rimbalzare
 - try/catch/finally
 - throw, throws
- Flussi e file
 - File, InputStream, FileInputStream, OutputStream, FileOutputStream, DataInputStream, DataOutputStream, RandomAccessFile

Stefano Mizzaro - API 2

46

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

47