Programmazione Orientata agli Oggetti in Linguaggio Java

Eccezioni: Gestione delle Eccezioni Parte b

versione 2.4

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Tipi di Eccezioni in Java
- Eccezioni Controllate
- La clausola finally

Tipi di Eccezioni in Java

- Tre tipi di eccezioni in Java
  - errori
  - eccezioni non controllate
  - eccezioni controllate
- Errori
  - tipicamente errori nel funzionamento della macchina virtuale; es: OutOfMemoryError
  - sono considerate normalmente irrecuperabili e non vengono catturate

Tipi di Eccezioni in Java

- Eccezioni non controllate
  - tipicamente dovute ad errori interni all'applicazione (errori logici o dell'utente)
  - es: IllegalArgumentException, IndexOutOfBoundsException, NumberFormatException
  - possono essere catturate o meno
  - sono spesso evitabili con una maggiore cura nella programmazione o da parte dell'utente

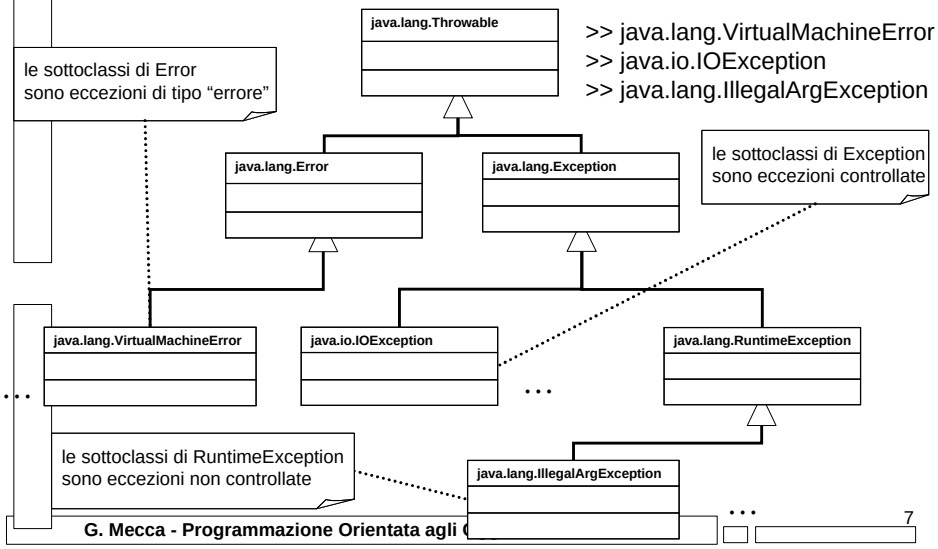
Tipi di Eccezioni in Java

- Eccezioni controllate
  - ⇨tipicamente dovute a problemi nella comunicazione con sistemi esterni
  - ⇨es: GeneralSecurityException, IOException, PrinterException, SQLException
  - ⇨sono considerate come effetti di situazioni da gestire necessariamente ma rimediabili
  - ⇨in questi casi il programma dovrebbe cercare di trovare una soluzione

Tipi di Eccezioni in Java

- Come distinguere
  - ⇨tra eccezioni controllate da eccezioni non controllate ?
  - ⇨sulla base delle loro “superclassi”
- La gerarchia delle eccezioni
  - ⇨tre classi principali: Throwable, Error, Exception e RuntimeException
  - ⇨il tipo dell’eccezione dipende dalla classe che estende

La Gerarchia delle Eccezioni



Tipi di Eccezioni in Java

- Il senso di "eccezione controllata"
  - ⇨in effetti "controllata" deve essere intesa come "controllata dal compilatore"
  - ⇨il compilatore effettua delle verifiche sul codice che coinvolge l'eccezione
  - ⇨e può generare errori sintattici sia relativi al chiamante che al chiamato

Tipi di Eccezioni in Java

>> Record.java

- In particolare
  - ⇨l'eccezione deve essere "dichiarata" nell'intestazione del metodo chiamato che può lanciarla, utilizzando la clausola "throws"
  - ⇨e non può essere ignorata dal chiamante
  - ⇨altrimenti si verificano errori sintattici

```
it\unibas\indovinasemplice\Record.java:42: unreported exception
java.io.FileNotFoundException; must be caught or declared to be thrown
```


Tipi di Eccezioni in Java

- Metafora
  - ⇨un'eccezione controllata è un messaggio per cui viene richiesta una "ricevuta di ritorno"
  - ⇨il chiamato richiede la ricevuta utilizzando la clausola throws nel prototipo
  - ⇨il chiamante non può ignorare l'eccezione (deve firmare la ricevuta di ritorno)
  - ⇨per esempio includendo la chiamata in una regione controllata (try-catch)

Eccezioni Controllate

- La clausola throws
  - segue il prototipo di un metodo ed elenca le eccezioni controllate che il metodo può lanciare
  - è obbligatoria per tutte le eccezioni controllate
  - opzionale per le altre eccezioni

Eccezioni Controllate

- Un esempio
  - il metodo getValoreRecord() della classe Record.java
- Varie eccezioni controllate
  - in java.io.FileReader:
    public FileReader(String fileName)
    throws java.io.FileNotFoundException
  - in java.io.BufferedReader:
    public String readLine()
    throws java.io.IOException

Eccezioni Controllate

- Attenzione
  - ⇨è possibile dichiarare anche eccezioni non controllate
- Esempio: in java.lang.Integer:
  - ⇨public static int parseInt(String val)
    throws java.lang.NumberFormatException
  - ⇨in questo caso l'eccezione entra a far parte del prototipo e quindi del "contratto" del metodo ma non viene controllata

Eccezioni Controllate

- Nel caso delle eccezioni controllate
  - ⇨l'eccezione deve essere necessariamente dichiarata dal chiamato
  - ⇨il chiamante ha due alternative
- Alternativa 1
  - ⇨catturare e gestire l'eccezione
- Alternativa 2
  - ⇨dichiararla a sua volta e lanciarla ulteriormente al suo chiamante

Nell'Applicazione: Alternativa 1

```
public int getValoreRecord() {
   int valoreRecord;
   java.io.BufferedReader file = null;
   try {
      file = new java.io.BufferedReader(
                  new java.io.FileReader(this.nomeFileRecord));
      String linea = file.readLine();
      valoreRecord = Integer.parseInt(linea);
   } catch (java.io.FileNotFoundException f) {
      valoreRecord = 101;
   } catch (java.io.IOException e) {
      System.out.println("Errore nella lettura: " + e);
      valoreRecord = 101;
   } catch (NumberFormatException nfe) {
      valoreRecord = 101;
   } ...
   return valoreRecord;
}
```


Alternativa 2

```
public int getValoreRecord()
        throws java.io.FileNotFoundException, java.io.IOException {
   int valoreRecord;
   java.io.BufferedReader file = null;
   file = new java.io.BufferedReader(
          new java.io.FileReader(this.nomeFileRecord));
   String linea = file.readLine();
   valoreRecord = Integer.parseInt(linea);
   return valoreRecord;
}
```

in questo caso le eccezioni non vengono sollevate direttamente nel metodo, ma il metodo dichiara di rilanciarle al suo chiamante nel caso vengano sollevate da readLine() o FileReader()

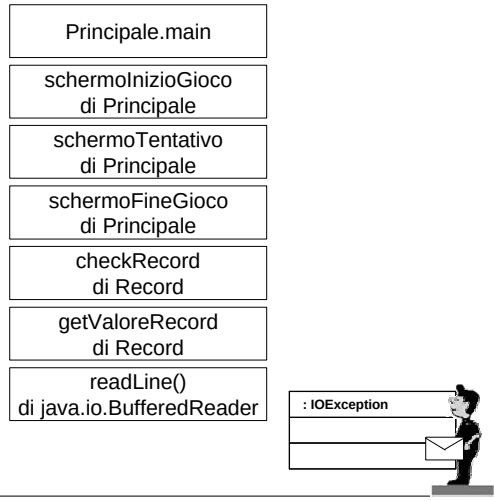
Eccezioni Controllate

- Attenzione
  - se getValoreRecord() rilancia l'eccezione, l'eccezione si "propaga" da un metodo all'altro
- Vediamo nel seguito i due casi
  - consideriamo una circostanza di utilizzo dell'applicazione
  - e valutiamo i due casi (a) eccezione catturata (b) eccezione rilanciata

Eccezioni Controllate

- Un esempio: Indovina il Numero
  - in Principale main chiama schermoInizioGioco()
  - schermoInizioGioco() chiama schermoTentativi()
  - schermoTentativi() chiama schermoFineGioco()
  - schermoFineGioco() chiama checkRecord() di Record
  - in Record.java checkRecord() chiama getValoreRecord()
  - getValoreRecord() chiama readLine() di java.io.BufferedReader
  - a questo punto si genera un'eccezione

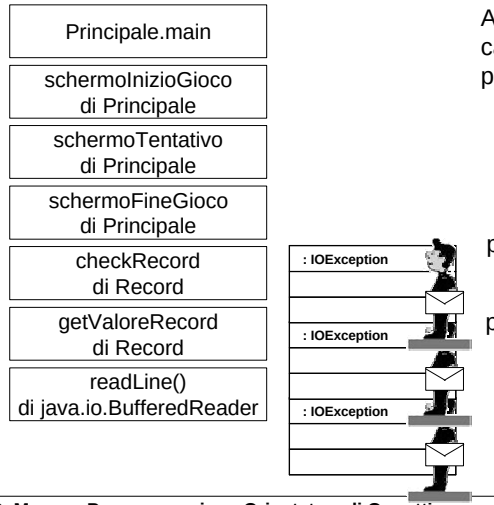
I Caso: L'Eccezione viene Catturata



NOTA: questo è effettivamente il caso della nostra applicazione

```
try {
    .....
} catch (IOException e) {
   record = 101;
}
```


II Caso: L'Eccezione viene Rilanciata



ATTENZIONE: in questo secondo caso il flusso di controllo è più intricato e meno controllabile

```
try {....} catch (IOException e) {
... }
public void checkValoreRecord(
  int x) throws IOException {....}
public void getValoreRecord()
  throws IOException { ....}
```

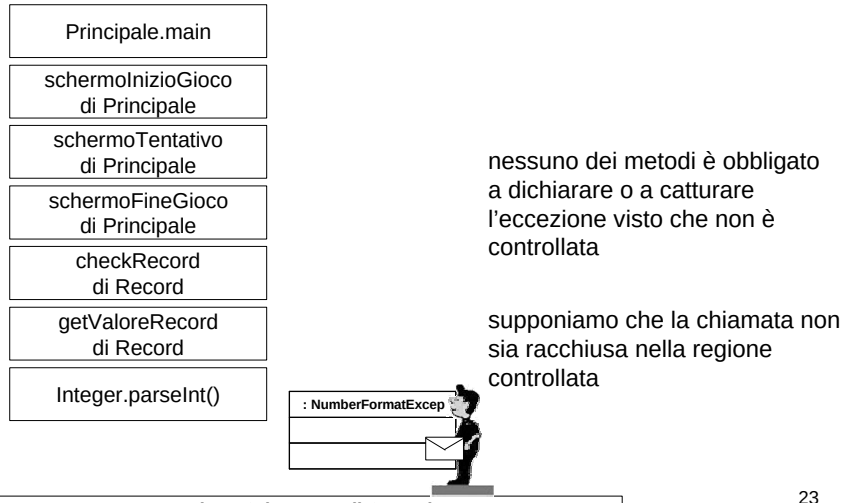
Eccezioni Controllate

- Nota
  - ⇨e se invece che IOException viene sollevata NumberFormatException da Integer.parseInt() ?
- Nel caso dell'applicazione
  - ⇨questa viene catturata dal blocco try-catch e viene subito gestita
- E se non ci fosse il blocco catch ?

```
public int getValoreRecord() {
   int valoreRecord;
   java.io.BufferedReader file = null;
   try {
      file = new java.io.BufferedReader(
                  new java.io.FileReader(this.nomeFileRecord));
      String linea = file.readLine();
      valoreRecord = Integer.parseInt(linea);
   } catch (java.io.FileNotFoundException f) {
      valoreRecord = 101;
   } catch (java.io.IOException e) {
      System.out.println("Errore nella lettura: " + e);
      valoreRecord = 101;
   } finally { ... }
   return valoreRecord;
}
```

NOTA: NumberFormatException è un'eccezione NON controllata; il blocco catch{...} è facoltativo e non è necessario dichiarare l'eccezione con la clausola throws

Una Ulteriore Alternativa



nessuno dei metodi è obbligato a dichiarare o a catturare l'eccezione visto che non è controllata

supponiamo che la chiamata non sia racchiusa nella regione controllata

Eccezioni Controllate

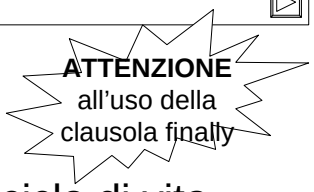
- Nel caso della classe Record
  - ⇨la prima soluzione viene adottata per evitare di propagare l'eccezione al controllo
- Linea guida
  - ⇨il controllo non dovrebbe avere percezione dei problemi sul file system
  - ⇨IOException piuttosto che DOMException piuttosto che SQLException
  - ⇨ci sono altre soluzioni per questo prob. (>>)

Eccezioni Controllate

- In altri casi
  - ⇨può essere opportuno non gestire e rilanciare l'eccezione al componente di controllo più adeguato a gestirla
- Esempio: la classe Console
  - ⇨non è opportuno in generale che la classe Console intervenga direttamente sull'interfaccia stampando messaggi
  - ⇨es: interfaccia con grafica a carattere

>>it.unibas.utilita.Console

La Clausola finally



- Utilizzo di risorse esterne: ciclo di vita
  - ⇨creazione (allocazione)
  - ⇨utilizzo
  - ⇨chiusura (rilascio)
- Esempio
  - ⇨flussi di lettura e scrittura da file
  - ⇨connessioni a DBMS
  - ⇨connessioni a URI

La Clausola finally

- In tutti questi casi
  - ⇨la chiusura è molto importante
  - ⇨spesso è necessaria per riportare la risorsa esterna in uno stato corretto
  - ⇨es: file o database
- Attenzione
  - ⇨la chiusura deve essere effettuata in tutti i casi, sia che l'utilizzo si svolga correttamente, sia che si verifichino eccezioni

La Clausola finally

- La clausola finally
  - ⇨in una regione controllata, consente di specificare un blocco di istruzioni di chiusura
  - ⇨il blocco viene eseguito in tutti i casi al termine dell'esecuzione della regione
  - ⇨al termine dell'esecuzione del blocco try, se non ci sono eccezioni
  - ⇨al termine dell'esecuzione del blocco catch, se c'è un'eccezione e viene catturata
  - ⇨subito dopo l'interruzione del blocco try se c'è un'eccezione e non viene catturata

```
private void setValoreRecord(int valoreRecord) {
      if (valoreRecord < 1) {
         throw new IllegalArgumentException("Numero di tentativi scorretto");
      }
      java.io.PrintWriter file = null;
      try {
         file = new java.io.PrintWriter(
                  new java.io.BufferedWriter(
                     new java.io.FileWriter(this.nomeFileRecord)));
         file.println(valoreRecord + "");
      } catch (java.io.IOException e) {
         System.out.println("Impossibile salvare il record: " + e);
      } finally {
         if (file != null) {
            file.close();
         }
      }
   }
```


La Clausola finally

- Nota
  - ⇨in alcuni casi, le operazioni del finally possono sollevare a loro volta eccezioni
- Esempio: getValoreRecord()

```
finally {
      try { // file è di tipo BufferedReader
         if (file != null) { file.close() }
      } catch (java.io.IOException ioe) { }
}
```

La Clausola finally

ATTENZIONE
MAI blocchi
catch vuoti !!!

- Attenzione: un grave errore
  - ⇨catturare un'eccezione e non fare niente
  - ⇨try {...} catch (IOException e) {}
  - ⇨in questo caso si nasconde un malfunzionamento dell'applicazione, di cui non resta traccia
  - ⇨il minimo da fare è stampare la descrizione dell'eccezione: System.out.println(e)

La Clausola finally

- Una delle poche eccezioni
  - ⇨la clausola finally
  - ⇨in questo caso il programmatore assume di avere fatto tutto il possibile per riportare la risorsa ad uno stato corretto
  - ⇨se questo non è possibile e si generano ulteriori eccezioni non c'è niente di significativo da fare

La Clausola finally

- Un'ultima annotazione
  - ⇨è consentito anche che ci siano blocchi try-finally senza catch
  - ⇨in quei casi in cui il chiamante non deve effettuare particolari operazioni sull'eccezione nel blocco catch
  - ⇨ma solo preoccuparsi di effettuare le operazioni finali di ripulitura

```
private void setValoreRecord(int valoreRecord)
          throws IOException {
    if (valoreRecord < 1) {
       throw new IllegalArgumentException("Numero scorretto");
    }
    java.io.PrintWriter file = null;
    try {
       file = new java.io.PrintWriter(
                 new java.io.BufferedWriter(
                    new java.io.FileWriter(this.nomeFileRecord)));
       file.println(valoreRecord + "");
    } finally {
       if (file != null) {
          file.close();
       }
    }
 }
```

in questo caso il blocco catch aveva l'unica funzione di stampare il messaggio dell'eccezione in alcuni casi potrebbe essere omesso

Riassumendo

- Tipi di Eccezioni in Java
- Eccezioni Controllate
- La clausola finally

Termini della Licenza