

# Linguaggi

Corso M-Z - Laurea in Ingegneria Informatica  
A.A. 2007-2008

Alessandro Longheu

<http://www.diit.unict.it/users/alongheu>

[alessandro.longheu@diit.unict.it](mailto:alessandro.longheu@diit.unict.it)

- lezione 11 -

## Gestione delle eccezioni in Java

1

A. Longheu – Linguaggi M-Z – Ing. Inf. 2007-2008

## Gestione degli errori

- Spesso vi sono istruzioni "critiche", che in certi casi possono produrre errori.
  - L'approccio classico consiste nell'inserire controlli (if... else..) per cercare di intercettare a priori le situazioni critiche
  - Ma è un modo di procedere spesso insoddisfacente
  - non è facile prevedere tutte le situazioni che potrebbero produrre l'errore
  - "gestire" l'errore spesso significa solo stampare a video un messaggio.
- Java introduce il concetto di eccezione
  - anziché tentare di prevedere le situazioni di errore, si tenta di eseguire l'operazione in un blocco controllato
  - se si produce un errore, l'operazione lancia un'eccezione
  - l'eccezione viene catturata dal blocco entro cui l'operazione è eseguita...
  - ... e può essere gestita nel modo più appropriato.

2



## Cos'è una eccezione?

---

- Errori causati da ...
- Esempi:
  - Un file che contiene informazioni errate
  - Il fallimento di una connessione di rete
  - Il fallimento di accesso ad un'unità di memoria di massa
  - Uso di indici invalidi in un array
  - L'uso di un riferimento ad un oggetto a cui non è stato assegnato un oggetto.

3



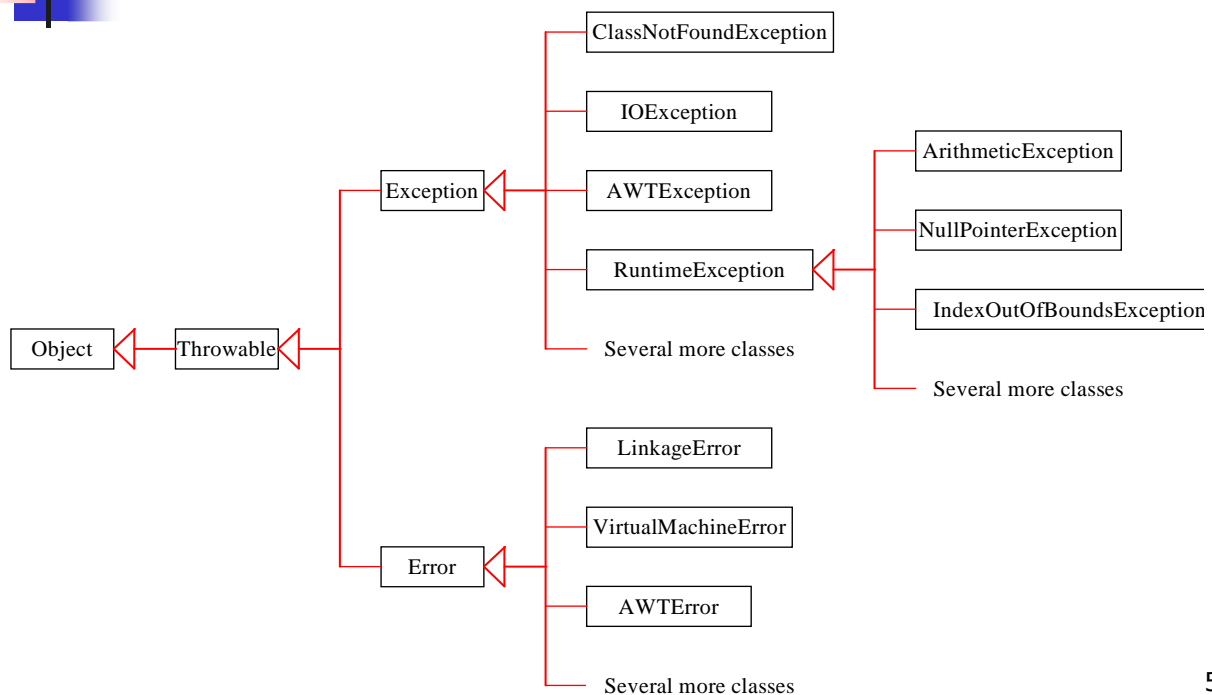
## Cos'è una eccezione?

---

- Una eccezione è un oggetto, istanza di Throwable o di una sua sottoclasse: le due sottoclassi tipiche sono Exception e Error
- Un Error indica problemi relativi al funzionamento della JVM e va solitamente considerato irrecuperabile: perciò non è da catturare, né da gestire (esempi: LinkageError, ThreadDeath,...)
- Una Exception indica invece situazioni recuperabili, almeno in linea di principio: va quindi catturata e gestita (esempi: fine file, indice di un array oltre i limiti, errori di input, etc.).
- Ci sono molti tipi di eccezioni predefinite in Java; i tipi delle eccezioni sono definiti come un tipo oggetto.
- È possibile definire le proprie eccezioni, estendendo Throwable o Exception (meglio perché più specifica, mentre Throwable comprende anche la classe Error); le eccezioni predefinite sono in realtà sufficienti a coprire una vasta casistica; il motivo che spinge alla creazione di nuove eccezioni è più che altro l'esigenza di fornire informazioni specifiche note al programmatore ed utili per il trattamento dell'eccezione stessa

4

## Tipi di eccezioni



5

## Controllo delle eccezioni

- Sebbene sia comunque utile e raccomandabile catturare e gestire le eccezioni, ciò non è obbligatorio per tutte le eccezioni
- RuntimeException (e derivate)
  - possono non essere gestite: per questo vengono dette non controllate.
  - È comunque buona norma gestirle; una eccezione non controllata può infatti propagarsi di blocco in blocco: se raggiunge il main senza essere stata catturata, il programma abortisce.

6

## Eccezione come oggetto

- Poiché un'eccezione è un oggetto, può contenere dati o definire metodi.
- Tutte le eccezioni definiscono un metodo `getMessage()` che restituisce il messaggio d'errore associato
  - alcune eccezioni definiscono anche dei campi dati (ad esempio, `bytesTransferred` in `InterruptedException`) che danno altre informazioni, utili per gestire la situazione.

7

## Classe Throwable

```

public Throwable()
    //Default constructor
    //Creates an instance of Throwable with an empty message string

public Throwable(String strMessage)
    //Constructor with parameters
    //Creates an instance of Throwable with message string specified
    //by the parameter strMessage

public String getMessage()
    //Returns the detailed message stored in the object

public void printStackTrace()
    //Method to print the stack trace showing the sequence of
    //method calls when an exception occurs

public void printStackTrace(PrintWriter stream)
    //Method to print the stack trace showing the sequence of
    //method calls when an exception occurs
    //Output is sent to the stream specified by the parameter stream.

public String toString()
    //Returns a string representation of the Throwable object
  
```

8

# Classe Exception

```
public Exception()
//Default constructor
//Creates a new instance of the class Exception

public Exception(String str)
//Constructor with parameters
//Creates a new instance of the class Exception. The parameter str
//specifies the message string
```

Exception Class	Description
ArithmeticException	Arithmetic errors such as division by 0
ArrayIndexOutOfBoundsException	Array index is either less than 0 or greater than or equal to the length of the array
FileNotFoundException	Reference to a file that cannot be found
IllegalArgumentException	Calling a method with illegal arguments
IndexOutOfBoundsException	An array index is out of bounds
NullPointerException	Reference to an object that has not been instantiated
NumberFormatException	Use of an illegal number format
StringIndexOutOfBoundsException	A string index is either less than 0 or greater than or equal to the length of the string

9

# Eccezioni sollevate da alcuni metodi

**Table 12-4** Exceptions Thrown by the Methods of the **class** Integer

<b>class</b> Integer		
Method	Exception Thrown	Description
parseInt(String str)	NumberFormatException	The string <b>str</b> does not contain an <b>int</b> value
valueOf(String str)	NumberFormatException	The string <b>str</b> does not contain an <b>int</b> value

**Table 12-5** Exceptions Thrown by the Methods of the **class** Double

<b>class</b> Double		
Method	Exception Thrown	Description
parseDouble(String str)	NumberFormatException	The string <b>str</b> does not contain a <b>double</b> value
valueOf(String str)	NumberFormatException	The string <b>str</b> does not contain a <b>double</b> value

10

# Eccezioni sollevate da alcuni metodi

**Table 12-6** Exceptions Thrown by the Methods of the `class` `String`

<code>class</code> <code>String</code>		
Method	Exception Thrown	Description
<code>String(String str)</code>	<code>NullPointerException</code>	<code>str</code> is <code>null</code>
<code>charAt(int a)</code>	<code>StringIndexOutOfBoundsException</code>	The value of <code>a</code> is not a valid index
<code>indexOf(String str)</code>	<code>NullPointerException</code>	<code>str</code> is <code>null</code>
<code>lastIndexOf(String str)</code>	<code>NullPointerException</code>	<code>str</code> is <code>null</code>
<code>substring(int a)</code>	<code>StringIndexOutOfBoundsException</code>	The value of <code>a</code> is not a valid index
<code>substring(int a, int b)</code>	<code>StringIndexOutOfBoundsException</code>	The value of <code>a</code> and/or <code>b</code> is not a valid index

**Table 12-7** Exceptions Thrown by the Methods of the `class` `StringTokenizer`

<code>class</code> <code>StringTokenizer</code>		
Method	Exception Thrown	Description
<code>StringTokenizer(String str)</code>	<code>NullPointerException</code>	<code>str</code> is <code>null</code>
<code>nextToken()</code>	<code>NoSuchElementException</code>	string is <code>null</code>

11

## Generazione (sollevamento) di eccezioni

- Quando si verifica un'anomalia essa può:
  - Essere causata da un'istruzione; in tal caso l'istruzione viene troncata; se ad esempio è un'espressione e l'eccezione è innescata dall'operando sinistro, il destro non sarà valutato
  - Essere generata esplicitamente con l'istruzione `throw <expr>`, che deve restituire un riferimento ad un oggetto `Throwable`:
 

```
if (x > 100)
    throw new Exception("x is too big");
```
  - Essere generata da un errore interno della JVM; in questo caso, nulla in genere si può fare e si parla di eccezione asincrona (le precedenti due sono di tipo sincrono, ovvero innescate come diretta conseguenza dell'esecuzione di un'istruzione)
- In ognuno dei tre casi, JVM lancia (throws) una nuova eccezione

12

# Gestione o rilancio delle eccezioni

- Al verificarsi dell'eccezione, si possono seguire due strade:
  - gestire l'eccezione, con un costrutto try / catch
  - rilanciarla esplicitamente all'esterno del metodo che contiene le istruzioni che hanno provocato l'eccezione, delegandone in pratica la gestione ad altri (unica eccezione: il main); se si sceglie questa seconda strada, il metodo deve indicare quali eccezioni possono "scaturire" da esso, con la clausola throws
- in assenza di tutto, costrutto try/catch e di clausola throws, viene invocato il gestore di default:
  - Mostra una stringa che descrive l'eccezione
  - Traccia il punto del programma in cui l'eccezione si è verificata
  - Termina il programma

13

# Costrutto try...catch

- Sintassi del costrutto:

```

try {
    statements . . .
} catch (ExceptionType1 ename1) {
    error handling statements . . .
} catch (ExceptionType2 ename2) {
    error handling statements . . .
} catch (ExceptionType3 ename3) {
    error handling statements . . .
} finally {
    statements . . .
}
  
```

14



## Costrutto try...catch

- Esempi:

```
try {
    readFromFile("datafile");
} catch (FileNotFoundException e) {
    System.err.println("Error: File not found");
}
```

```
try {
    readFromFile("datafile");
} catch (Exception e) {
    System.err.println("Error: " + e);
}
```

15



## Costrutto try...catch

- Deve essere presente almeno un catch o finally; Se l'operazione lancia diversi tipi di eccezione in risposta a diversi tipi di errore, più blocchi catch possono seguire lo stesso blocco try
- il corpo del blocco try viene eseguito fino al momento in cui viene sollevata un'eccezione da un'istruzione; a quel punto viene esaminata ogni clausola catch per vedere se il tipo dell'eccezione sollevata coincide con quella trattata dal blocco catch; se esiste un siffatto catch, viene inizializzato l'identificatore (il parametro enamexx) e vengono eseguite le istruzioni; nessuna altra catch verrà eseguita
- se non viene trovata nessuna catch adatta, l'eccezione è propagata al di fuori del try, ad altri try più esterni se presenti
- se presente una finally, le sue istruzioni sono eseguite comunque, anche in assenza di eccezioni; la finally è solitamente usata per attuare e garantire la coerenza dello stato degli oggetti (ad esempio, chiudere file)

16





## Costrutto try...catch

- l'ordine delle catch è rilevante; se una catch con un'eccezione X viene posta prima della catch di una Y e X è una superclasse di Y, X intercetta sempre anche le eccezioni che dovrebbero spettare a Y, rendendo la catch di Y irraggiungibile, il che non è probabilmente quello che si desidera; il compilatore segnala errore in tali situazioni
- try intercetta solo un'eccezione; se se ne verificasse un'altra a causa delle istruzioni della catch attualmente in esecuzione, essa non viene presa in considerazione; può intervenire eventualmente in tal caso una try esterna

17



## Costrutto try...catch

- Esempio di conversione stringa/numero: la conversione è svolta dal metodo statico `int Integer.parseInt(String s)`
- L'operazione è critica, perché può avvenire solo se la stringa data contiene la rappresentazione di un intero; Se ciò non accade, `parseInt` lancia una `NumberFormatException`

```

class EsempioEccezione {
    public static void main(String args[]){
        int a = 0;
        String s = "1123";
        try { a = Integer.parseInt(s); }
        catch (NumberFormatException e) {
            System.out.println("Stringa mal fatta");
        }
    }
}

```

18



## Costruito try...catch

- Altro esempio:

```
try {
    int value = 0;
    int answer = 100 / value;
    System.out.println("Questa riga non verrà stampata");
} catch (Exception e) {
    System.out.println("Errore");
}
System.out.println("Il programma continua...")
```

19



## Costruito try...catch

- Altro esempio:
 

```
for (int i = 0; i < 2; i++)    System.out.println(args[i]);
System.out.println("Fine!");
```
- se non vengono passati parametri o ne viene passato uno solo, viene lanciata una eccezione
- Il tipo di eccezione viene stampato
- La chiamata a println("Fine!") non è mai eseguita. Un errore è lanciato e il programma finisce.
- Meglio scrivere:
 

```
try {
    for (int i = 0; i < 2; i++)    System.out.println(args[i]);
} catch (Exception e) {
    System.out.println("Uso di un numero di parametri errato ...");
}
System.out.println("Fine! ");
```

20

## Costrutto try...catch

- Esempio di eccezioni multiple:

```
try {
    int r1 = Integer.parseInt(str);
    int r2 = r1 / x;
} catch (NumberFormatException e) {
    System.out.println("Formato errato!");
} catch (Exception e) {
    System.out.println("");
}
System.out.println("Fine!");
```

- Cosa accade per le seguenti stringhe di input:
  - `str = "10" and x = 5?`
  - `str = "abc" and x = 5?`
  - `str = "10" and x = 0?`

21

## Costrutto try...catch

- Le eccezioni si propagano lungo la ordinaria catena delle chiamate  
*public void method1(String str) {*

```
    try {
        System.out.println("Method1 Start");
        int r1 = Integer.parseInt(str);
    } catch (ArithmeticException e) {System.out.println("Calculation Error!");}
    System.out.println("Method1 End");
}

public static void main(String args[]) {
    try {
        method1(args[0]);
    } catch (ArrayIndexOutOfBoundsException e) {System.out.println("Out of
    Index Error!");}
    } catch (Exception e) {    System.out.println("General Error!");}
    System.out.println("Finished!");
}
```

22

## Costrutto try...catch

- Esempio di finally:

```
try {
    String a = "0",
        int r2 = Integer.parseInt(a) / Integer.parseInt(a);
} catch (ArithmeticException e) {
    System.out.println("Calculation Error");
} catch (Exception e) {
    System.out.println("General Exception");
} finally {
    System.out.println("Finally");
}
System.out.println("Finished");
```

- L'esecuzione produce un'eccezione e le stringhe:

```
Calculation Error
Finally
Finished
```

23

## Costrutto try...catch

- Esempio di finally:

```
try {
    String a = "100",
        int r2 = Integer.parseInt(a) / Integer.parseInt(a);
} catch (ArithmeticException e) {
    System.out.println("Calculation Error");
} catch (Exception e) {
    System.out.println("General Exception");
} finally {
    System.out.println("Finally");
}
System.out.println("Finished");
```

- L'esecuzione è senza eccezioni e produce le stringhe:

```
Finally
Finished
```

24

## Costruito try...catch

- Esempio di finally:
 

```
try {
    String c = "abcde",
        int r1 = Integer.parseInt(c);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Error!");
    } catch (ArithmeticException e) {
        System.out.println("Calculation Error!");
    } finally { System.out.println("Finally block!");
    }
    System.out.println("Finished!");
```
- L'esecuzione produce l'eccezione non intercettata:

```
Finally block!
Exception in thread "main" java.lang.NumberFormatException: abcde
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at Except7_3.main(Except7_3.java:5)
```

25

## Rilancio di eccezioni

- In alternativa al costruito try...catch, si può rilanciare l'eccezione indicandola con la clausola throws

```
try {
    // set up a FileReader
    FileReader f = new FileReader("in.txt");
    } catch (FileNotFoundException e) {
        System.out.println("File not found");
    }
}
```

- oppure

```
public static void main(String[] args) throws
FileNotFoundException {
    // set up a FileReader
    FileReader f = new FileReader("in.txt");
}
```

26



## Rilancio di eccezioni

---

- La clausola `throws` specifica un elenco di eccezioni (controllate) sollevabili da un dato metodo o classe
- le eccezioni non controllate sono la classe `RuntimeException` e le sue sottoclassi, la classe `Error` e le sue sottoclassi
- il metodo può sollevare le eccezioni indicate nella clausola ma anche sottoclassi delle stesse; se un metodo solleva una eccezione `X` tuttavia è buona norma indicare la stessa `X` nella `throws`, piuttosto una generica `Z` superclasse di `X` che consente una maggiore generalità ma fa perdere il dettaglio su `X`, apportando un'eccessiva astrazione alla semantica del codice
- in caso di overriding di un metodo con `throws` da parte di una sottoclasse della classe ove il metodo è definito, la `throws` del metodo ridefinente deve mantenere la compatibilità, potendo ossia dichiarare eccezioni sottoclassi di quelle indicate nella `throws` del metodo originario, senza aggiungere altre eccezioni 27




## Analisi di eccezioni

---

- Il metodo `PrintStackTrace()` è utilizzato per determinare l'ordine con cui il metodo viene chiamato e le eccezioni gestite

# Analisi di eccezioni



```

public class PrintStackTraceExample2
{
    public static void main(String[] args)
    {
        methodA();
    }

    public static void methodA()
    {
        try
        {
            methodB();
        }
        catch(Exception e)
        {
            System.out.println(e.toString() + " caught in methodA");
            e.printStackTrace();
        }
    }

    public static void methodB() throws Exception
    {
        methodC();
    }

    public static void methodC() throws Exception
    {
        throw new Exception("Exception generated in method C");
    }
}

```

29

# Analisi di eccezioni

## Output

```

java.lang.Exception: Exception generated in method C caught in main
java.lang.Exception: Exception generated in method C
    at PrintStackTraceExample1.methodC
    (PrintStackTraceExample1.java:31)
    at PrintStackTraceExample1.methodB
    (PrintStackTraceExample1.java:26)
    at PrintStackTraceExample1.methodA
    (PrintStackTraceExample1.java:22)
    at PrintStackTraceExample1.main
    (PrintStackTraceExample1.java:11)

```

30



## Utilizzo di eccezioni ed asserzioni

---

- L'eccezione modella una situazione inattesa; non si dovrebbe ricorrere pertanto alle eccezioni per individuare situazioni prevedibili (ad esempio, fine del file)
- un altro strumento offerto da Java per controllare lo stato di esecuzione del programma è rappresentato dalle asserzioni, ossia condizioni che devono essere verificate altrimenti provocano un Error (no Exception)
- la sintassi per un'asserzione è: `assert <expr> [:dettaglio]`
- le asserzioni dovrebbero essere usate insieme alle eccezioni (non in alternativa), in quanto il loro scopo è segnalare situazioni di stato assolutamente inconsistente, laddove le eccezioni possono rappresentare situazioni ancora meno critiche (trattabili e che potrebbero permettere l'avanzamento del programma)

31



## Utilizzo di eccezioni ed asserzioni

---

- le asserzioni possono essere attivate o meno; di default sono disattivate;
- data la disattivabilità, un'asserzione non deve interferire con il codice, ad esempio `assert ++i < max;` è molto pericolosa perché non è detto che `++i` sia sempre eseguita
- le asserzioni possono essere usate in fase di development dell'applicazione perché ne raffinano il debug, disattivandole in fase di rilascio; un approccio robusto potrebbe comunque suggerire di mantenerle attive anche dopo il rilascio, per scovare eventuali bug durante l'utilizzo da parte degli utenti

32