

# Progettazione del Software

## Programmazione in Java (3)

Le eccezioni

Domenico Fabio Savo

Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti

**Sapienza Università di Roma**  
**Anno Accademico 2017/2018**

Le slide di questo corso sono il frutto di una rielaborazione di analogo materiale redatto da Marco Cadoli, Giuseppe De Giacomo, Maurizio Lenzerini e Domenico Lembo

---

# Eccezioni

---

- L'esistenza di una eccezione indica che si è verificato un problema durante l'esecuzione di un programma (situazione anomala)
- Questa è una situazione che un metodo non è in grado di controllare (ma per la quale si possono prendere provvedimenti opportuni)
- Java possiede un meccanismo che permette al programmatore di trattare le situazioni anomale in modo flessibile e perfettamente integrato con la metodologia orientata ad oggetti
- **le eccezioni sono viste come oggetti di classi particolari**

---

# Propagazione delle eccezioni

---

Come funziona il meccanismo di gestione delle eccezioni?

- Quando si verifica un imprevisto, il metodo attivo lancia (**throws**) un'eccezione che viene passata al metodo chiamante
- Il metodo attivo termina l'esecuzione (come con return)
- Per default, un metodo che riceve un'eccezione termina l'esecuzione e passa l'eccezione al metodo chiamante
- Quando l'eccezione raggiunge il main, l'esecuzione del programma termina stampando un opportuno messaggio di errore

---

# Comportamento di default (esempio)

---

Per sperimentare con la propagazione delle eccezioni, si compili e si esegua la classe `NestedNullPointerException`:

```
1  public class NestedNullPointerException {
2      public static void bar(){
3          Object o = null;
4          System.out.println(o.toString());
5      }
6      public static void foo(){
7          bar();
8      }
9      public static void main(String [] args){
10         foo();
11     }
12 }
```

---

# Comportamento di default (esempio)

---

La macchina astratta Java scriverà qualcosa come:

```
> java NestedNullPointerException
Exception in thread "main" java.lang.NullPointerException
at NestedNullPointerException.bar(NestedNullPointerException.java:4)
at NestedNullPointerException.foo(NestedNullPointerException.java:7)
at NestedNullPointerException.main(NestedNullPointerException.java:10)
```

elencando la catena dei metodi attivi nel momento in cui si verifica l'eccezione (bar - foo - main) e per ogni metodo la linea di codice dove si è verificata.

---

# Le istruzioni `try-catch-finally`

---

È possibile gestire le eccezioni in modo che il programma non termini in modo “disastroso” è il seguente

- Il programmatore racchiude in un blocco `try` il codice che può generare una eccezione
- Il blocco `try` è immediatamente seguito da zero o più blocchi `catch`
- Un blocco `catch` specifica i tipi di eccezioni che può gestire, e il codice che le gestisce
- Un blocco `finally` **facoltativo**, dopo l'ultimo blocco `catch`, specifica del codice che viene **sempre** eseguito

---

# Le istruzioni try-catch-finally

---

```
try
{ /* blocco try */ }

catch(TipoEcc1 e1)
{ /* blocco catch */ }
catch(TipoEcc2 e2)
{ /* blocco catch */ }
...
catch(TipoEcc2 e2)
{ /* blocco catch */ }

finally
{ /* blocco finally */ }
```

---

# Le istruzioni `try-catch-finally`

---

- Quando viene lanciata una eccezione, il programma **abbandona il blocco** `try` e ricerca il gestore appropriato nei blocchi `catch`
- Se il tipo di eccezione lanciata corrisponde a quello di un blocco `catch`, allora il codice di quel blocco viene eseguito, l'esecuzione riprende dopo l'ultimo blocco `catch`
- Se non sono lanciate eccezioni nel blocco `try`, l'esecuzione riprende dopo l'ultimo blocco `catch`
- Se c'è un blocco `finally`, questo viene **sempre** eseguito



---

# Le istruzioni try-catch-finally (esempio)

---

```
public class NestedNullPointer2 {
    public static void bar(){
        Object o = null;
        try{ System.out.println(o.toString()); }
        catch(NullPointerException e)
            { System.out.println("Si e' verificata una eccezione di "+
                "tipo NullPointerException"); }
        finally { System.out.println("Questo viene sempre stampato"); }
    }
    public static void foo(){
        bar();
    }
    public static void main(String [] args){
        foo();
    }
}
```

---

## Comportamento di default (esempio 2)

---

L'eccezione può essere catturata in uno qualsiasi dei metodi in cui è propagata

```
public class NestedNullPointer3 {
    public static void bar(){
        Object o = null;
        System.out.println(o.toString());
    }
    public static void foo(){
        try{ bar(); }
        catch(NullPointerException e)
            { System.out.println("Si e' verificata una eccezione di "+
                "tipo NullPointerException nel metodo bar()"); }
        finally { System.out.println("Questo viene sempre stampato"); }
    }
    public static void main(String [] args){
        foo();
    }
}
```

---

# Nota

---

Le eventuali istruzioni del blocco `finally` vengono eseguite sempre, anche in presenza di una eccezione verificatasi nel blocco `try` che non viene catturata da alcun blocco `catch`, o in presenza di un `return` del blocco `try` o `catch`. Il blocco `finally` può contenere delle istruzioni che chiudono dei files oppure rilasciano delle risorse, per garantire la consistenza dello stato.

---

# La clausola throws

---

Le eccezioni che non sono state trattate tramite le istruzioni `try-catch-finally` vanno dichiarate nella clausola `throws`, facente parte del prototipo del metodo in cui l'eccezione si può verificare

```
// File Esempio5.java
import java.io.*;

public class Esempio5 {
    public static void main(String[] args) throws IOException {
        // stampa su schermo il file passato tramite linea di comando
        FileInputStream istream = new FileInputStream(args[0]);
        BufferedReader in = new BufferedReader(new InputStreamReader(istream));
        String linea = in.readLine();
        while(linea != null) {
            System.out.println(linea);
            linea = in.readLine();
        }
        in.close();
    }
}
```

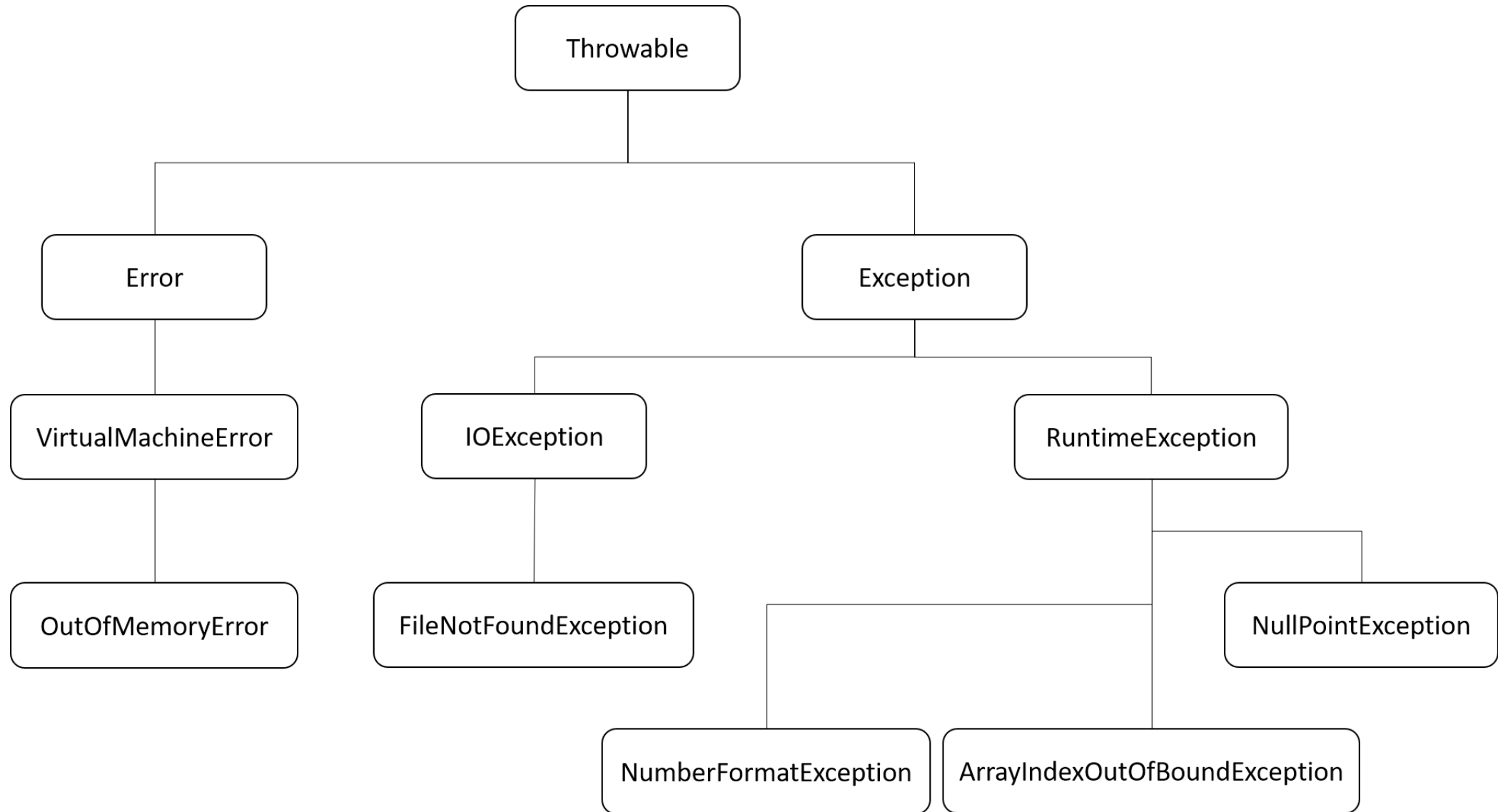
---

# Eccezioni checked vs unchecked

---

- In realtà, **non è necessario specificare nella clausola `throws` tutti i tipi di eccezione non “catturate”**
- Non si ha quest’obbligo per le eccezioni che ereditano dalla classe `RuntimeException` (**unchecked**)
- Lo stesso dicasi per gli errori, che sono oggetti che ereditano dalla classe `Error`; questi oggetti rappresentano errori di sistema che non devono essere gestiti
- In pratica, eccezioni di tipo `RuntimeException` non devono necessariamente essere gestite, e cioè non devono necessariamente essere catturate con il meccanismo `try-catch`, o dichiarate con la clausola `throws`
- Si noti che nella classe `NestedNullPointerException` presentata all’inizio di queste slide, l’eccezione che si può verificare è di tipo `NullPointerException`, che è una sottoclasse di `RuntimeException`, e per questo non deve essere necessariamente gestita (come fatto nella classe `NullPointerException2`)

# Gerarchia delle eccezioni



---

# Definire una propria classe di eccezioni

---

È possibile farlo estendendo la classe `Exception` (o una sua sottoclasse). Ad esempio:

```
public class MyException extends Exception {  
  
    private String messaggio;  
  
    public MyException(String m) {  
        messaggio = m;  
    }  
  
    public String toString() {  
        return messaggio;  
    }  
}
```

---

# Come lanciare una eccezione

---

- Per lanciare una eccezione si usa l'istruzione `throw`, che accetta un qualunque oggetto `Throwable`
- Si possono anche definire proprie classi di eccezioni, derivandole da `Exception`

```
MyException e;
```

```
...
```

```
throw e;
```

Le eccezioni lanciate vanno segnalate nella clausola `throws` (ovviamente questo non è necessario se si tratta di `RuntimeException`)



---

# Esempio

---

```
public class Divisione {
    public static float divisione(float x, float y) throws MyException{
        if ( y==0 )
            throw new MyException("La divisione per zero non e' possibile");
        return x/y;
    }

    public static void main(String [] args){
        int a=3, b=0;
        try { System.out.println(divisione(a,b)); }
        catch (MyException e)
            { System.out.println(e.toString());}
    }
}
```