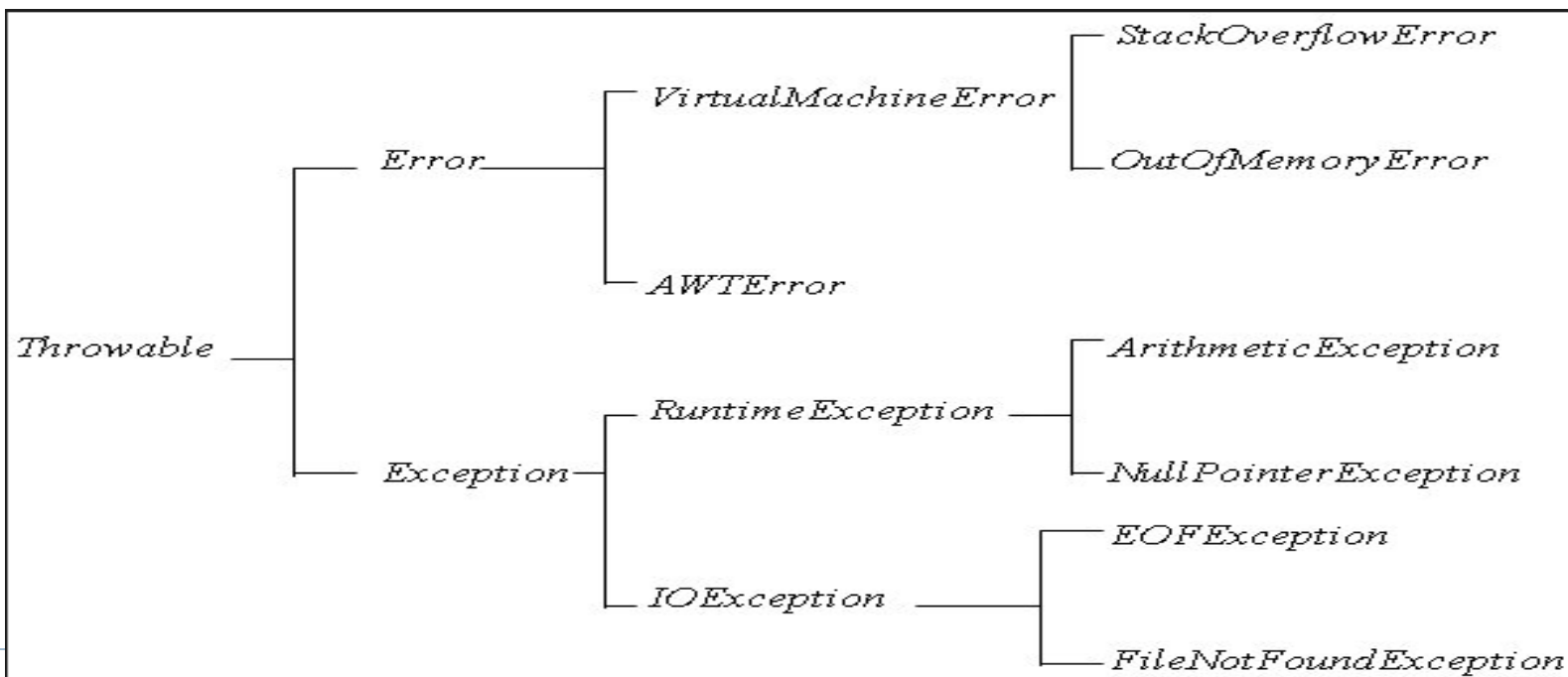

Le eccezioni in Java

di Roberta Molinari

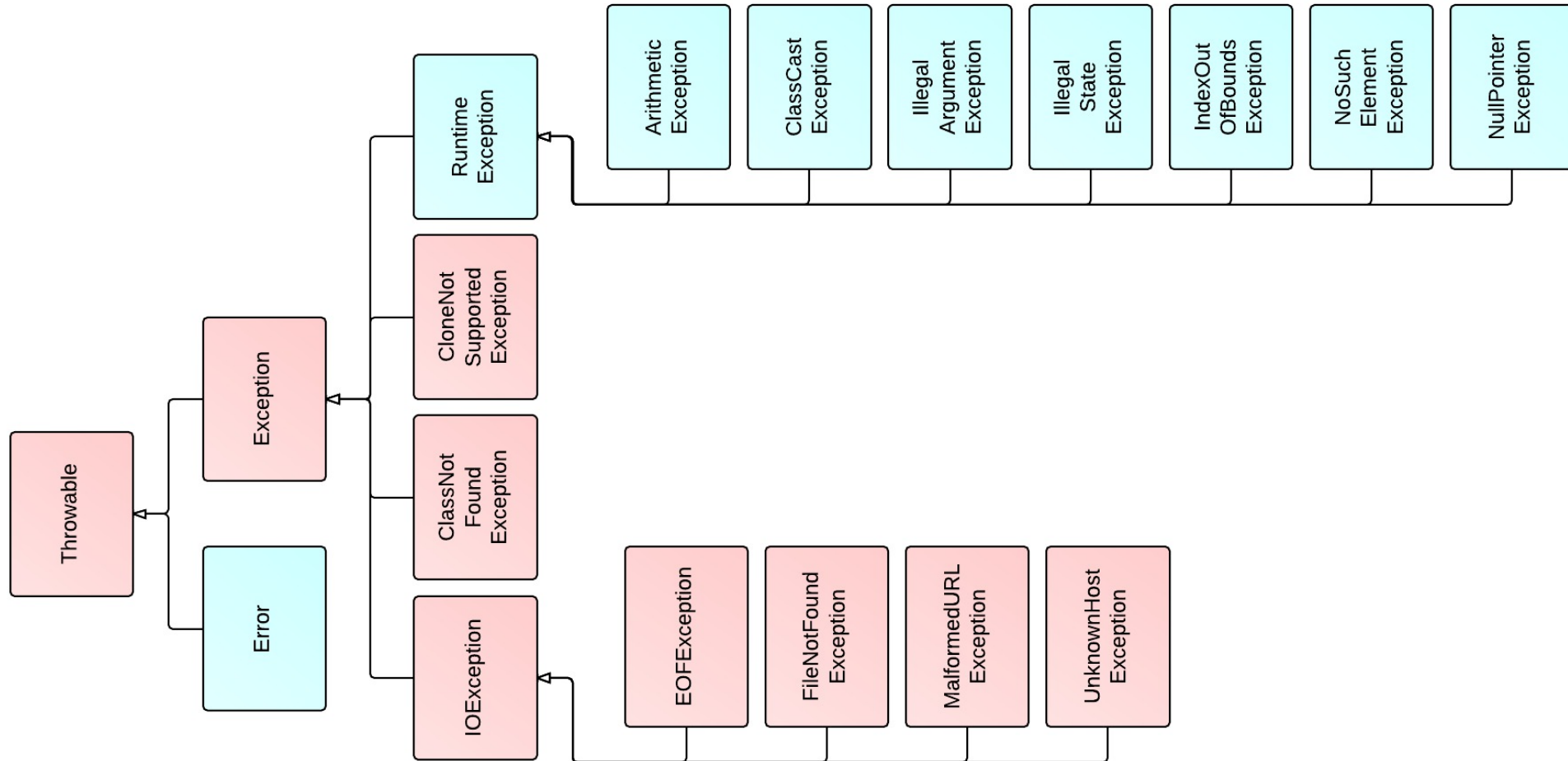
Le eccezioni

Sono situazioni anomale che si verificano durante l'esecuzione di un programma, vengono generate da un'istruzione e in alcuni casi possono essere raccolte e gestite da altre parti del programma. Se non gestite si segnala un errore e il programma termina. Le eccezioni della classe *Error* non sono recuperabili.

I possibili tipi di eccezione derivano dalla classe **Throwable**



Gerarchia delle eccezioni



Le eccezioni

Un'**Exception** in Java è un oggetto che descrive una situazione anomala o un errore recuperabile. Possono essere:

► **controllate**

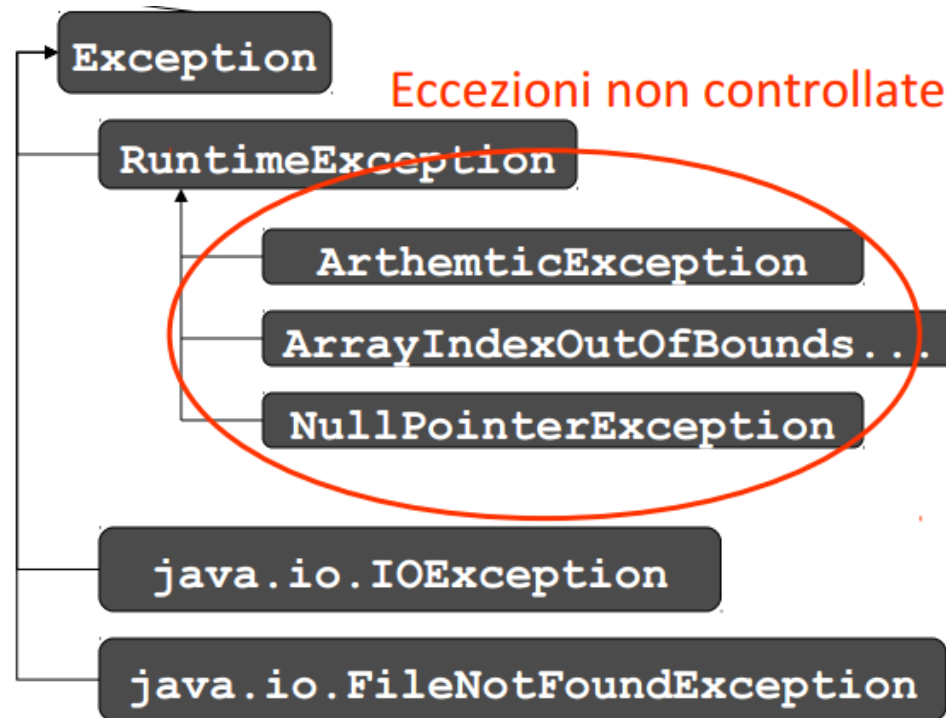
- dovute a eventi esterni al programma, es: cercare di accedere ad un file inesistente
- si chiamano controllate perché il compilatore controlla che vengano esplicitamente indicate e intercettate

► **non controllate**

- dovute all'esecuzione del programma e che potrebbero essere evitate con codice robusto (non sono controllate dal compilatore perché imprevedibili).

Le eccezioni non controllate

- ▶ Non richiedono una gestione esplicita
- ▶ Discendono da **RuntimeException** o da una sua classe discendente



Le eccezioni controllate

- ▶ Un'eccezione controllata deve essere intercettata da un metodo in una clausola `catch` e quindi gestita (**handle**) o deve essere dichiarata (**declare**) nella lista delle clausole `throws` di ciascun metodo che possa lanciare l'eccezione o propagarla
 - Un metodo che può sollevare un'eccezione controllata, che quindi non gestisce, deve dichiararlo con la clausola **throws** (il compilatore segnala se un'eccezione controllata non viene gestita propriamente)
 - A sua volta un metodo che lo richiama deve gestirla o dichiararla, cioè deve:
 - gestire l'eccezione con **try ... catch** oppure
 - dichiarare a sua volta che potrà sollevare l'eccezione nella clausola **throws** e così la propaga
- ▶ Le eccezioni posseggono il metodo `.printStackTrace()` che stampa la traccia dello stack nel momento in cui si è verificata l'eccezione e l'attributo `message` che contiene il messaggio di errore impostato per quella eccezione

Gestione dell'eccezioni

try..catch

```
try{
    //se si generano eccezioni si passa al blocco catch
}
catch (tipo1 | tipo2 e){
    //si indica quale/i eccezione/i può verificarsi e cosa fare
    //si possono elencare eccezioni che non siano derivate
}
[catch (tipo3 e3){//se gestisce in modo diverso i vari tipi di
    possibili eccezioni
}]
[finally{    //operazioni da fare comunque con o senza
    eccezioni. Se in un catch c'è una throw prima di lanciare
    quella nuova esegue prima la finally
}]
```

Quando si genera un'eccezione sarà "catturata" dal primo catch di cui è istanza: se metto prima `catch (Exception e)` i successivi catch non verranno mai catturati perché relative a sue sottoclassi!!

Le eccezioni finally

- Il blocco **finally** solitamente usato per stampare messaggi per debugging del codice, lanciare messaggi personalizzati in caso di eccezioni, chiudere oggetti come il **BufferedReader**, chiudere le sessioni aperte ed altro.

```
BufferedReader br = null;
try{
    /*
     * creazione del BufferedReader e suo utilizzo
     */
} catch(IOException e) {
    e.printStackTrace();
} finally{
    if(br != null){
        br.close(); // chiusura del canale
    }
}
```

Le eccezioni

try-with-resources

- ▶ Il **try-with-resources** (introdotta da Java 7) serve per semplificare la chiusura di **oggetti che continuano ad occupare inutilmente risorse** se non vengono chiusi.
- ▶ Le istanze di classi che implementano **l'interfaccia `java.lang.AutoCloseable`** sono indicate come **risorse**. Queste in genere devono essere smaltite subito, senza aspettare il garbage collector.
- ▶ L'interfaccia `AutoCloseable` definisce un solo metodo:

```
public void close() throws Exception
```
- ▶ Si richiede che il metodo generi un'eccezione più specifica di `Exception`, o nessuna eccezione.

Le eccezioni

try-with-resources

- ▶ Una vasta gamma di classi e interfacce Java standard implementano **AutoCloseable**:
 - ▶ **InputStream** , **OutputStream** e le loro sottoclassi
 - ▶ **Reader** , **Writer** e le loro sottoclassi
 - ▶ **Socket** e **ServerSocket** e relative sottoclassi
 - ▶ **Channel** e le sue sottoclassi, e
 - ▶ il JDBC interfaccia **Connection** , **Statement** e **ResultSet** e le loro sottoclassi

Le eccezioni

try-with-resources

```
try      (BufferedReader      br      =      new
  BufferedReader(new FileReader(path)) ) {
    /* codice vario */
} catch(IOException e) {
    e.printStackTrace();
}
```

- ▶ Una volta che le variabili della risorsa sono state inizializzate, viene eseguito il blocco try.
- ▶ `br.close()` verrà chiamato automaticamente, al termine del blocco indipendentemente dalla sua evoluzione, proprio come se fosse stato usato un blocco try-catch-finally.

Lanciare un'eccezione

throw

L'eccezioni vengono lanciate dall'interprete in fase di run-time quando si verifica una situazione imprevista. Si può lanciare un'eccezione (controllata o non) anche da programma con il comando **throw**.

```
throw    objEcc;    //istanza    di    una    sottoclasse  
           di Exception
```

più comunemente

```
throw    new    ExceptionClass (msgErrore) ;
```

Definire e gestire eccezioni definite dal programmatore

Un utente può definire una sua classe di eccezioni e usarle come quelle di Java (può lanciarle, catturarle, delegarne la gestione). Le eccezioni di una nuova classe sono *controllate* (IOException o anche solo Exception) oppure *non controllate* (RuntimeException) a seconda della superclasse.

```
class Mia_Exception extends Exception{ //controllate
    public Mia_Exception(String s){
        super(s);    //s= messaggio di errore
    }
    public Mia_Exception(){
        super("Messaggio predefinito");
    }
}
```

Definire e gestire eccezioni definite dal programmatore

Una volta definite possono essere lanciate:

```
void mio_metodo() throws Mia_Exception {  
...  
    if (cond)  
        throw new Mia_Exception("Mess errore");  
... }
```

e catturate nel chiamante (o propagate con throws)

```
try{  
    obj.mio_metodo();  
} catch (Mia_Exception e) {  
    System.out.println(e.getMessage());  
}
```