CORSO JAVA AESYS
Parte Prima

2022

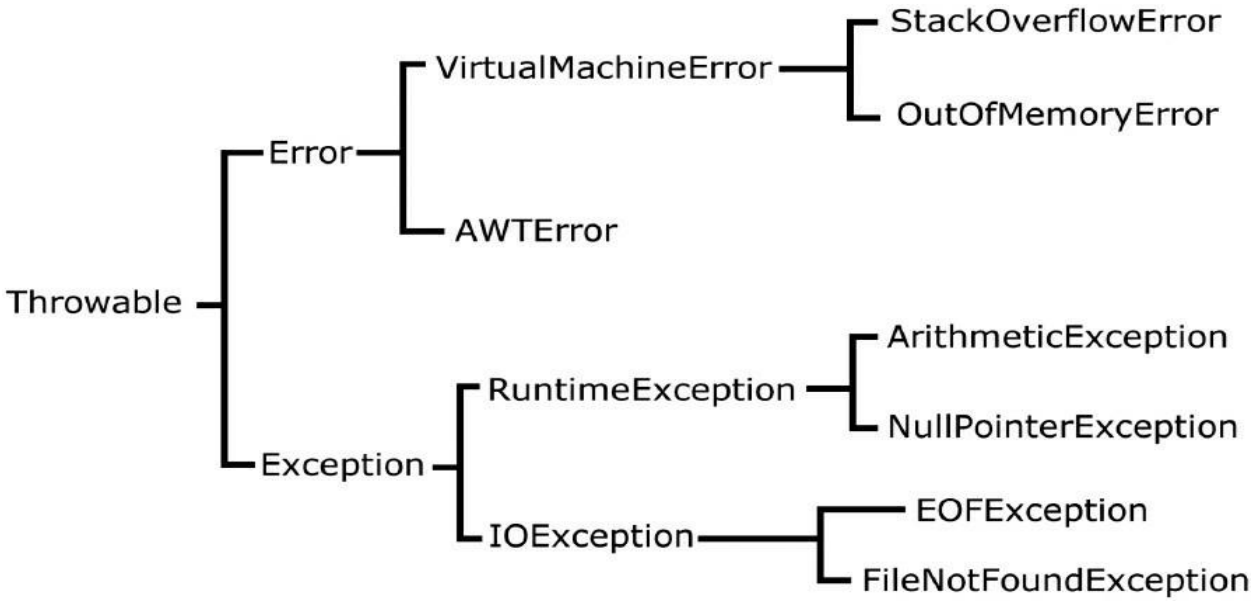
www.aesystech.it

Eccezioni, errori

- Gerarchie e categorizzazioni
- Meccanismo per la gestione delle eccezioni
- Try with resources
- Eccezioni personalizzate e propagazione dell’eccezione
- Warnings

Gerarchie e categorizzazioni

Nella libreria standard di Java esiste una gerarchia di classi che mette in relazione la classe Exception e la classe Error. Entrambe queste classi estendono la superclasse Throwable, come si può vedere in figura .



Un'ulteriore categorizzazione delle eccezioni è data dalla divisione delle eccezioni in checked ed unchecked exception. Ci si riferisce alle RuntimeException (e alle sue sottoclassi) come unchecked exception. Tutte le altre eccezioni (ovvero tutte quelle che non estendono RuntimeException), sono dette checked exception. Se si utilizza un metodo che lancia una checked exception senza gestirla da qualche parte, la compilazione non andrà a buon fine. Da qui il termine checked exception (in italiano "eccezioni controllate").

Meccanismo per la gestione delle eccezioni 1/2

lo sviluppatore ha a disposizione alcune parole chiave per gestire le eccezioni: try, catch, finally, throw e throws. Se bisogna sviluppare una parte di codice che potenzialmente può scatenare un'eccezione, è possibile circondarla con un blocco try seguito da uno o più blocchi catch. Per esempio:

```
public class Ecc1 {    public static void main(String args[]) {       int a = 10;       int b =
0;       int c = a/b;
      System.out.println(c);
   }
```

Questa classe può essere compilata senza problemi, ma genererà un'eccezione durante la sua esecuzione, dovuta all'impossibilità di eseguire una divisione per zero. In tal caso la JVM, dopo aver interrotto il programma, produrrà il seguente output:

Exception in thread "main" java.lang.ArithmeticException: / by zero at Ecc1.main(Ecc1.java:6)

L'unico problema è che il programma è terminato prematuramente. Utilizzando le parole chiave try e catch sarà possibile gestire l'eccezione in maniera personalizzata, facendo terminare il programma senza che sia interrotto bruscamente:

Meccanismo per la gestione delle eccezioni 2/2

Quando la JVM eseguirà tale codice incontrerà la divisione per zero della prima riga del blocco try e lancerà l'eccezione ArithmeticException, che verrà catturata nel blocco catch seguente. Quindi non sarà eseguita la riga che doveva stampare la variabile c, bensì quella che stampa la stringa
"Divisione per zero..." con la quale abbiamo gestito l'eccezione ed abbiamo permesso al nostro programma di terminare in maniera naturale. Come il lettore avrà sicuramente notato la sintassi dei blocchi try - catch è piuttosto strana, ma presto ci si fa l'abitudine, perché è presente più volte praticamente in tutti i programmi Java. In particolare il blocco catch deve dichiarare un parametro (come se fosse un metodo) del tipo dell'eccezione che deve essere catturata. Nell'esempio precedente il reference exc puntava proprio all'eccezione che la JVM aveva istanziato e lanciato. Infatti, tramite esso è possibile reperire informazioni proprio sull'eccezione stessa. Il modo più semplice per ottenere informazioni su ciò che è successo è invocare il metodo printStackTrace() sull'eccezione.

```
public class Ecc2 {    public static void main(String args[]) {
int a = 10;        int b = 0;        try {            int c = a/b;
          System.out.println(c);
      }
      catch (ArithmeticException exc) {
          System.out.println("Divisione per zero...");
      }
   }
}
```

Try with resources

alla versione 7 di Java, alcune classi ed interfacce (tra cui Connection) sono state riviste per supportare il meccanismo del cosiddetto try with resources. Questo consente la chiusura automatica degli oggetti che necessiterebbero di essere chiusi, una volta utilizzati.
La sintassi prevede la dichiarazione dell'oggetto (o degli oggetti) da chiudere automaticamente come parametri del blocco try. Segue un esempio (equivalente al precedente):

```
try(Connection conn =
  DriverManager.getConnection(url, username, password);
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM PERSONA
")) {

    while (rs.next()) {
        System.out.println(rs.getString(1));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Gli oggetti chiudibili conn, stmt e rs, sono quindi dichiarati come se fossero parametri del blocco try (che per l'occasione dovrebbe essere chiamato blocco try with resources). Quando il blocco terminerà la sua esecuzione, le tre risorse verranno automaticamente chiuse. Questo avverrà sia nel caso positivo (il codice viene eseguito correttamente), sia nel caso negativo (per esempio viene lanciata un'eccezione). Esattamente come se i comandi per chiudere le risorse si trovassero all'interno di una clausola finally. Si tratta di un vantaggio non da poco

Eccezioni personalizzate e propagazione dell'eccezione

per un particolare programma, potrebbe essere una eccezione anche una divisione per 5. Più verosimilmente, un programma che deve gestire in maniera automatica le prenotazioni per un teatro potrebbe voler lanciare un'eccezione nel momento in cui si tenti di prenotare un posto non più disponibile. In tal caso la soluzione è estendere la classe Exception ed eventualmente aggiungere membri e effettuare override di metodi come toString(). Segue un esempio:

```
public class PrenotazioneException extends Exception {     public
PrenotazioneException() {
        // Il costruttore di Exception chiamato inizializza la
        // variabile privata message
        super("Problema con la prenotazione");
    }
    public String toString() {        return getMessage() + ": posti esauriti!";
    }
}
```

La nostra eccezione contiene informazioni sul problema e rappresenta una astrazione corretta. Tuttavia la JVM non può lanciare automaticamente una
PrenotazioneException nel caso si tenti di prenotare un posto non più disponibile. La JVM, infatti, sa quando lanciare una ArithmeticException ma non sa quando lanciare una PrenotazioneException. In tal caso sarà compito dello sviluppatore lanciare l'eccezione. Esiste infatti la parola chiave throw (in inglese "lancia") che permette il lancio di un'eccezione tramite la seguente sintassi:

```
throw new PrenotazioneException();
```

AESYS
colors your business

Warnings

Quando compiliamo un file Java, il compilatore oltre a compilare correttamente o dare errori, potrebbe anche restituire dei warning (in italiano “avvertimento”). Per esempio, compilando la classe PrenotazioneException (e in generale una qualsiasi classe che estende Throwable) con un IDE qualsiasi otterremo il seguente output:

```
PrenotazioneException.java:1: warning: [serial] serializable class
  PrenotazioneException has no definition of serialVersionUID public class
PrenotazioneException extends Exception {
      ^
1 warning
```

Non si tratta di un errore, la classe è stata compilata e il file .class è stato creato. Il compilatore ci sta solo avvertendo che questo file è stato definito in modo potenzialmente pericoloso. In particolare ci è stato segnalato che la nostra classe è serializzabile e non ha definito un serialVersionUID

AESYS
colors your business