



# Gestione degli errori

Modulo 226A

Implementare in base alle classi (senza ereditarietà)

# Introduzione

Nei programmi possono verificarsi errori e situazioni impreviste.

## Esempi:

- Si cerca di accedere ad un elemento di un array che non esiste (fuori dai limiti)
- Si cerca di interpretare una stringa come un numero intero tramite il metodo **parseInt**, ma la stringa non contiene la rappresentazione di un numero intero.

Come dobbiamo gestire tali errori?

# Esempio

```
public static void main(String[] args) {  
    int n = 5;  
    int risultato = n / 0; //ArithmeticException: / by zero  
}
```

# Exception

Le eccezioni sono un meccanismo per gestire situazioni anomale durante l'esecuzione dei metodi.

La gestione delle eccezioni deve garantire i seguenti principi:

- le eccezioni non devono poter essere trascurate
- le eccezioni devono poter essere gestite da un gestore competente, non semplicemente dal chiamante del metodo che fallisce.

Questo implica che una eccezione, quando occorre, deve poter attraversare più di un metodo fino a che non ne trova uno in grado di gestirla.

# Exception

Java definisce che, quando una regola semantica viene violata, la JVM lancia (throw) un'eccezione e l'esecuzione viene trasferita dal punto del programma dove si è verificato l'errore ad un punto specificato dal programmatore nel quale tale eccezione viene gestita o catturata.

Esempi di eventi che sollevano un'eccezione:

- Una divisione per 0
- Il parsing/conversione di una stringa non convertibile
- L'accesso ad un indice inesistente in un array
- Il tentativo di accesso ad un file inesistente.

# Esempi di exception

Di seguito sono elencati alcuni tipi specifici di eccezione Java:

- **ArithmeticException:** causata da un'operazione aritmetica non consentita
- **NumberFormatException:** sollevata quando si tenta di convertire una stringa non appropriata in numero.
- **ArrayIndexOutOfBoundsException:** provocata dal tentativo di accesso ad un elemento inesistente di un array.
- **NullPointerException:** sollevata quando un programma utilizza un riferimento nullo invece di un oggetto.
- **Exception:** superclasse di tutte le eccezioni.

# Esempio

```
public static void main(String[] args) {  
    int elementi[] = {5,3,5};  
  
    int n = elementi[4]; //ArrayIndexOutOfBoundsException: Index 3 out of bounds for length
```

```
public static void main(String[] args) {  
    int n = 5;  
    int risultato = n / 0; //ArithmeticException: / by zero  
}
```

# Gestione

Quando all'interno di un metodo avviene un errore, l'esecuzione del metodo viene arrestato e la JVM solleva (**throws**) un'eccezione (**try/catch**)

Il try/catch permette di gestire localmente le eccezioni.

```
public static void main(String[] args) {  
    int n = 5;  
    int risultato = 0;  
  
    try{  
        risultato = n/0;  
    }catch(ArithmeticException ae){  
        System.out.println("Divisione per zero non ammessa");  
        System.out.println("Inserire un divisore maggiore diverso da 0");  
    }  
}
```



# Blocco try

Le istruzioni che possono generare delle eccezioni vengono scritte all'interno del blocco try.

```
...  
  
try{  
    risultato = n/0;  
}  
  
...
```

# Blocco catch

L'**exception handler** è costituito dal blocco catch.

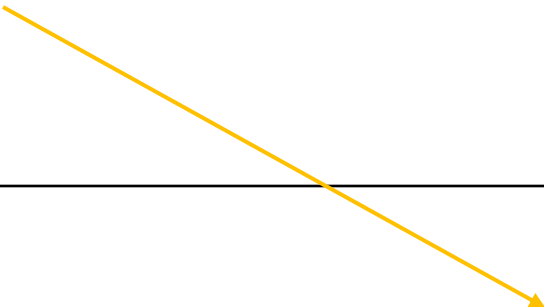
Nel blocco viene scritto il codice di gestione di una o più eccezioni specifiche, ossia le istruzioni da eseguire per gestire un errore.

L'oggetto dichiarato tra le parentesi tonde del blocco rappresenta l'eccezione da gestire

```
...  
  
}catch(ArithmeticException ae){  
    System.out.println("Divisione per zero non ammessa");  
    System.out.println("Inserire un divisore maggiore diverso da 0");  
}  
...
```

# Informazioni

Tale **oggetto** contiene tutte le informazioni relative all'errore avvenuto e dei metodi utili per manipolare tali informazioni.



```
...  
  
}catch(ArithmeticException ae){  
    System.out.println("Valore non valido" + ae.getMessage());  
}  
...
```

# Blocco finally

Il codice inserito nel blocco opzionale **finally** viene eseguito in ogni caso, sia che l'esecuzione del blocco **try** avvenga correttamente, sia che provochi un'eccezione.

```
public static void main(String[] args) {  
    int r = 0;  
    int a = 0;  
    try {  
        System.out.println("Inizio divisione");  
        r = 30 / a;  
        System.out.println("\tRisultato: " + r);  
    } catch (ArithmeticException ae) {  
        System.out.println("\tErrore: " + ae.getMessage());  
    } finally {  
        System.out.println("Fine divisione");  
    }  
}
```

Solitamente le azioni inserite nella clausola **finally** sono quelle relative al rilascio di risorse esterne, oppure alla chiusura di un file dopo averlo aperto.

# Catch multipli

Se le eccezioni da gestire sono più di una, è possibile aggiungere un gestore di eventi (catch).

```
try{
    int x = Integer.parseInt(args[0]);
    int y = 1 / x;
}catch(NumberFormatException nfe){//se il valore non è un numero
    System.out.println("Valore non valido!!");
}catch(ArithmeticException ae){//se la divisione provoca un errore
    System.out.println("Errore durante la divisione!!");
}
```

# Catch multipli

In alternativa è possibile utilizzare la sintassi **Multicatch**.

```
try {  
    int x = Integer.parseInt("1");  
    int y = 1 / x;  
} catch (NumberFormatException | ArithmeticException e) {  
    System.out.println("Errore !!");  
}
```

# Eccezioni e call stack

Quando all'interno di un metodo avviene un errore, l'esecuzione viene arrestata e la JVM solleva (throws) un'eccezione.

A questo punto il sistema di runtime di Java cerca un gestore per l'eccezione (exception handler).

Un exception handler è costituito da un blocco di codice (una serie di istruzioni) che il programmatore ha previsto per gestire un certo errore specifico.

Il gestore per l'eccezione viene cercato nei metodi che sono stati invocati, partendo dal metodo in cui si è verificata l'eccezione e risalendo a ritroso il call stack.

# Esempio

Nell'esempio seguente il metodo principale invoca il metodo **metodoA()** che a sua volta invoca **metodoB()**, il quale invoca il metodo **metodoC()**.

Quando viene invocato il **metodoC()** esso fa una divisione e stampa il risultato.

```
public class Esempio {  
    public static void main(String[] args) {  
        metodoA(0);  
    }  
    public static void metodoA(int n) {  
        metodoB(n);  
    }  
    public static void metodoB(int n) {  
        metodoC(n);  
    }  
    public static void metodoC(int n) {  
        int result = 1 / n;  
        System.out.println("Risultato " + result);  
    }  
}
```

Si verifica un'eccezione di tipo **ArithmeticException** causata dal tentativo di eseguire la divisione per zero (1/0).

Quando ciò avviene il sistema di **runtime** inizia a cercare un **exception handler** percorrendo il call stack a ritroso.

La ricerca avverrà nel modo seguente:



```
Exception in thread "main"  
java.lang.ArithmeticException: / by zero  
    at Esempio.metodoC(Esempio.java:16)  
    at Esempio.metodoB(Esempio.java:12)  
    at Esempio.metodoA(Esempio.java:8)  
    at Esempio.main(Esempio.java:4)
```



# Checked Unchecked

Le eccezioni si dividono in due classi:

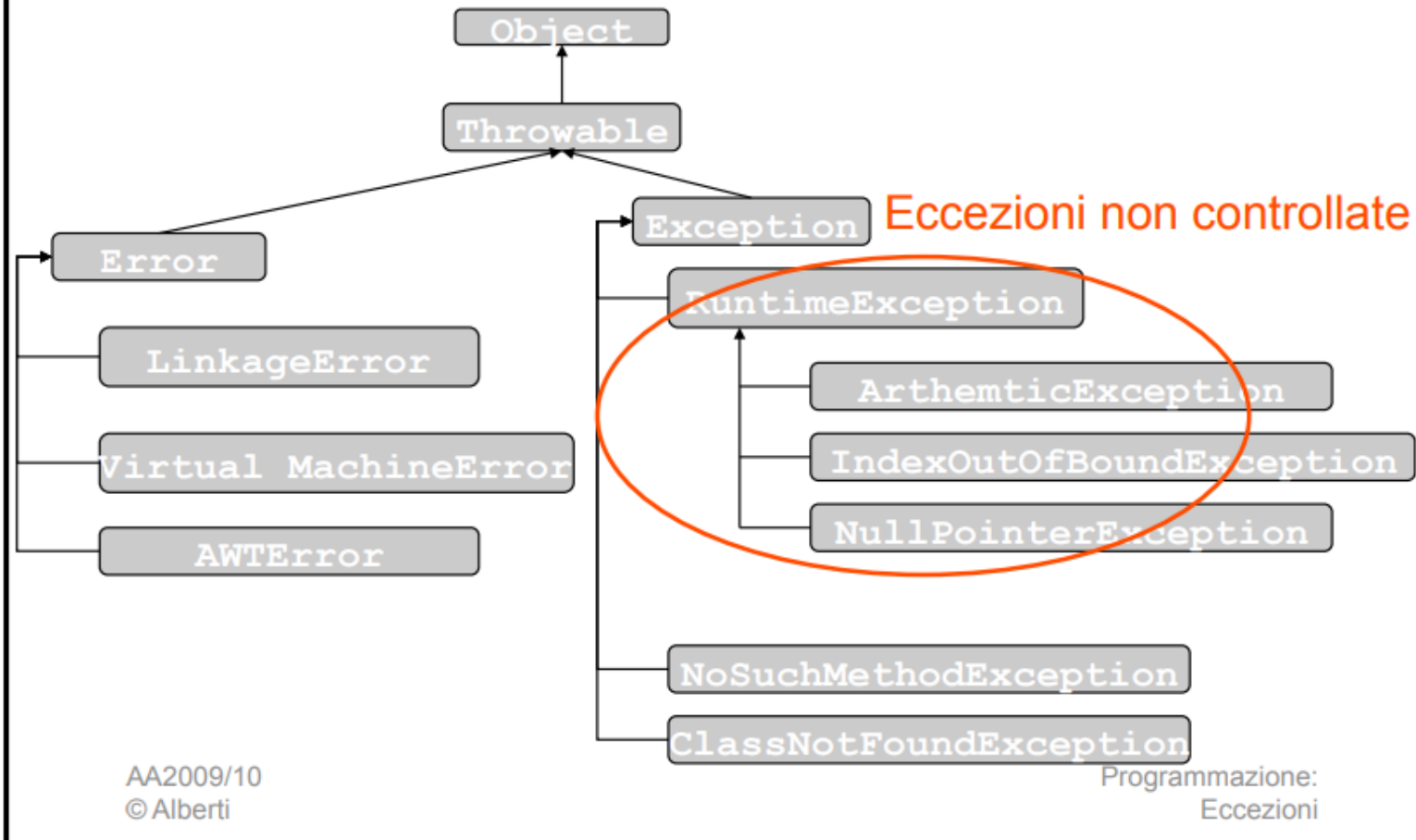
- Eccezioni **controllate** (*checked*)
- Eccezioni **non controllate** (*unchecked*)

Le eccezioni **controllate** devono essere gestite esplicitamente dal programma. Se un'istruzione *può* lanciare un'eccezione controllata, allora l'istruzione deve essere in un blocco **try-catch** che gestisce quel tipo di eccezione; oppure il metodo che contiene l'istruzione deve **delegare** la gestione al chiamante, con la clausola **throws**.

# Esempio

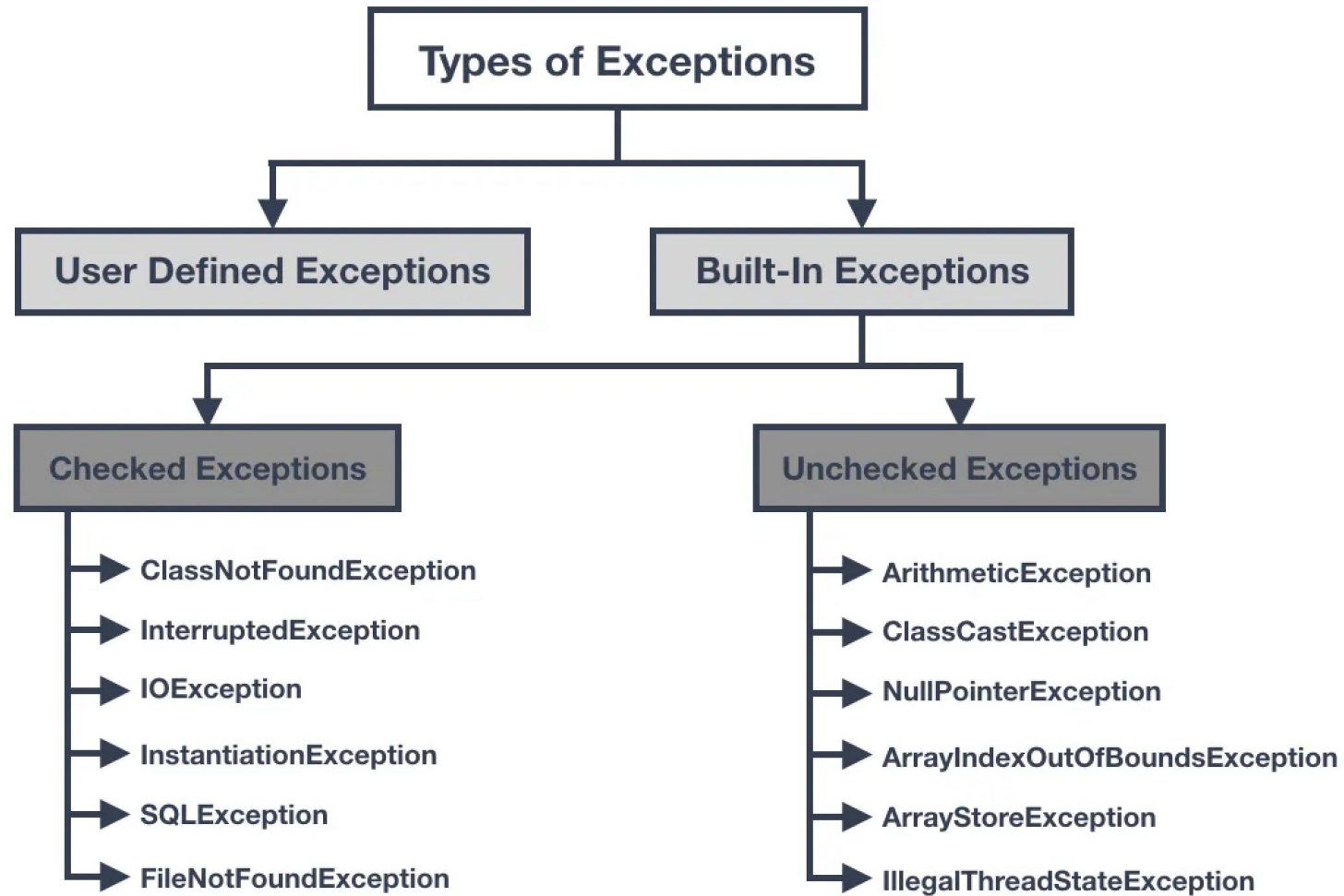
```
public class GestioneNonLocale {  
    public static void main(String[] args){  
        Stampa();  
    }  
    static public void Stampa(){  
        LeggiFile();  
    }  
    static public String LeggiFile(){  
        FileInputStream file = new FileInputStream("dizionario.txt");  
        //FileInputStream legge un flusso di byte  
        return "Fine lettura";  
    }  
}
```

# La gerarchia di classe delle eccezioni



Le eccezioni non controllate (Unchecked) sono quelle che si verificano a runtime.

# Checked



# throws

La gestione locale delle eccezioni (all'interno di un metodo) non è obbligatoria.

Infatti, se si verifica un errore in un metodo che non contiene un blocco try catch appropriato, il gestore appropriato verrà cercato automaticamente a ritroso nel call stack.

In questi casi è però importante che i programmatori che invocano il metodo sappiano che tale metodo potrebbe sollevare una certa eccezione.

La parola riservata throws serve proprio a dichiarare quali eccezioni vengono potenzialmente sollevate ma non vengono gestite da un metodo.

In questo modo le eccezioni potranno essere gestite a monte.

# Esempio in classe

Creare un programma che verifica se un numero (passato come argomento) è pari o dispari.

# Soluzione

```
public static void main(String[] args) {  
    int n = 0;  
    try{  
        n = Integer.parseInt(args[0]);  
        if(n % 2 == 0){  
            System.out.println(n + " è pari");  
        }  
        else{  
            System.out.println(n + " è dispari");  
        }  
    }catch(NumberFormatException nf){  
        System.out.println("Inserire solo numeri");  
    }  
}
```