

# LE ECCEZIONI

Dott. PhD Denis Ferraretti

*denis.ferraretti@unife.it*

## Che cosa sono le eccezioni

In un programma si possono generare situazioni critiche che provocano errori.

Non è però accettabile, soprattutto in applicazioni complesse, che un qualunque errore possa bloccare e far terminare in modo anomalo un programma.

Le situazioni di errore devono essere gestite.

☒ **Le eccezioni sono lo strumento messo a disposizione da Java per gestire in modo ordinato le situazioni anomale.**

## Esempio

Scriviamo un semplice programma che converte in numero una stringa passata sulla riga dei comandi:

```
public class EsempioEccezione
{
    public static void main(String args[])
    {
        int a = 0;
        String s = args[0];
        a = Integer.parseInt(s);
    }
}
```

Se la stringa passata non contiene un numero ci troviamo di fronte ad una situazione critica.

 **Il programma termina con un errore!**

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Prevenire è meglio che curare...

La soluzione “classica” consiste nel cercare di prevenire la situazione di errore inserendo controlli:

```
public class EsempioEccezione
{
    public static boolean isNumeric(String s)
    {
        boolean ok = true;
        for(int i=0; i<s.length(); i++)
            ok = ok && (Character.isDigit(s.charAt(i)));
        return ok;
    }

    public static void main(String args[])
    {
        int a = 0;
        String s = args[0];
        if (isNumeric(s))
            a = Integer.parseInt(s);
    }
}
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## ...o forse no

In situazioni semplici un approccio di questo tipo può funzionare.  
Ma in generale non è una soluzione efficace!

Infatti:

- In situazioni complesse i possibili errori sono molti e non si riesce ad individuarli e prevenirli tutti.
- I test sono spesso complessi da realizzare.
- I test devono essere eseguiti anche quando tutto va bene e questo può creare problemi di prestazioni.

☒ **Sarebbe quindi preferibile poter gestire gli errori solo quando si verificano.**

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## la soluzione JAVA

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 **solleva un'eccezione**.

L'eccezione viene **catturata** dal blocco entro cui l'operazione è eseguita e può essere **gestita** nel modo più appropriato.

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Gestire un'eccezione

La soluzione corretta in Java è quella di inserire le istruzioni “a rischio” in un blocco controllato:

```
public class EsempioEccezione
{
    public static void main(String args[])
    {
        int a = 0;
        String s = args[0];
        try
        {
            a = Integer.parseInt(s);
        }
        catch (Exception ex)
        {
            a = 0;
        }
    }
}
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Blocchi try/catch

Un blocco controllato è costituito da una clausola **try** e da una o più clausole **catch**:

```
try
{
    /* operazione critica */
}
catch (Exception ex)
{
    /* gestione dell'eccezione */
}
```

Nel blocco **try** inseriamo le istruzioni che possono generare situazioni di errore.


Se tutto va bene il blocco try viene eseguito e si passa all'istruzione successiva al blocco catch.

Se si verifica un'eccezione l'esecuzione del blocco try termina e si passa al blocco **catch** dove si può intervenire per gestire correttamente l'anomalia.


STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Flusso delle eccezioni

Riprendiamo il nostro esempio e vediamo cosa accade nei due casi:



```
s = "123";  
try  
{  
    a = Integer.parseInt(s);  
}  
catch (Exception ex)  
{  
    a = 0;  
}  
a = a + 2;  
  
// Il valore di a è 125
```



```
s = "xyz";  
try  
{  
    a = Integer.parseInt(s);  
}  
catch (Exception ex)  
{  
    a = 0;  
}  
a = a + 2;  
  
// Il valore di a è 2
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Ancora sul flusso

💣 **Attenzione:** se in un blocco try abbiamo più istruzioni quando si verifica un'eccezione le istruzioni successive non vengono eseguite:

```
s = "xyz";  
try  
{  
    a = Integer.parseInt(s);  
    a = a + 5; // Non viene eseguita!  
}  
catch (Exception ex)  
{  
    a = 0;  
}  
// dopo il catch il flusso riprende qui sotto  
a = a + 2 // a vale 2 e non 7!
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Che cos'è un'eccezione?

## ☑ Una eccezione è un oggetto

E' un' istanza di `java.lang.Throwable` o di una sua sottoclasse.

Le due sottoclassi più comuni sono `java.lang.Exception` e `java.lang.Error`

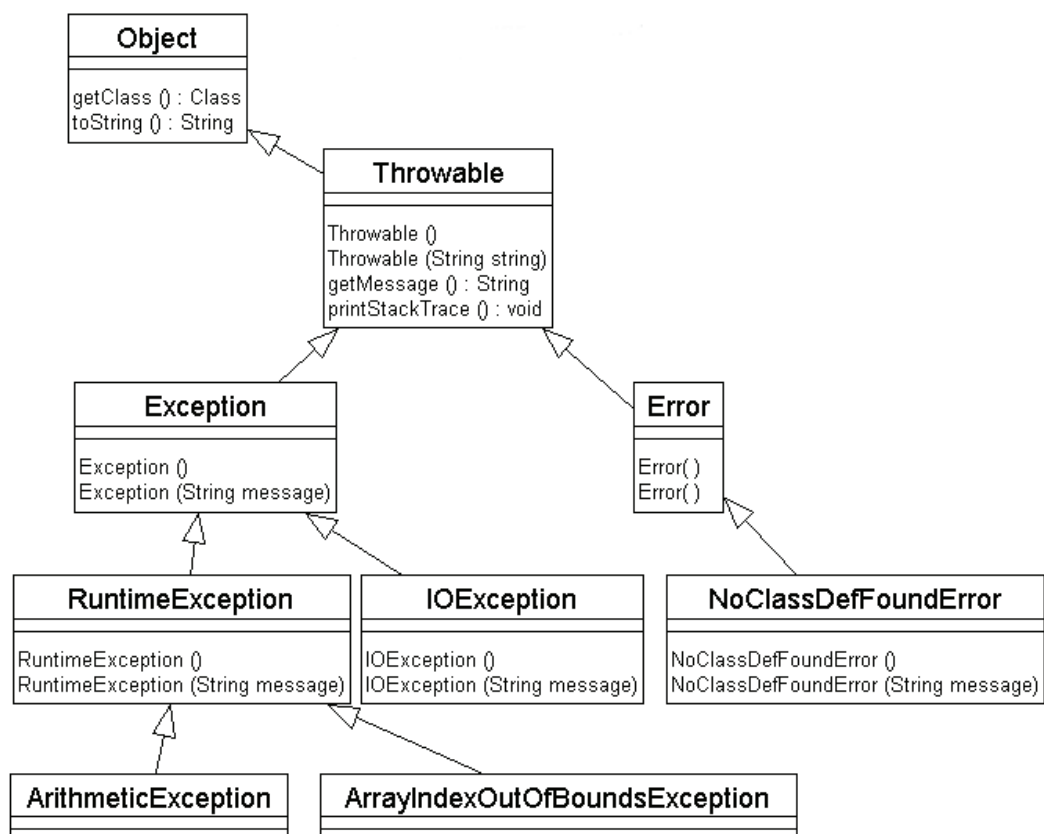
La parola "eccezione" è però spesso riferita a entrambe ma c'è una differenza:

- Un **Error** indica un grave problema di sistema, normalmente irrecoverabile: quindi non deve essere gestito.
- Una **Exception** indica invece una situazione recuperabile: dovrebbe essere gestita.

Nei casi di nostro interesse abbiamo quindi a che fare con istanze di sottoclassi di `java.lang.Exception`

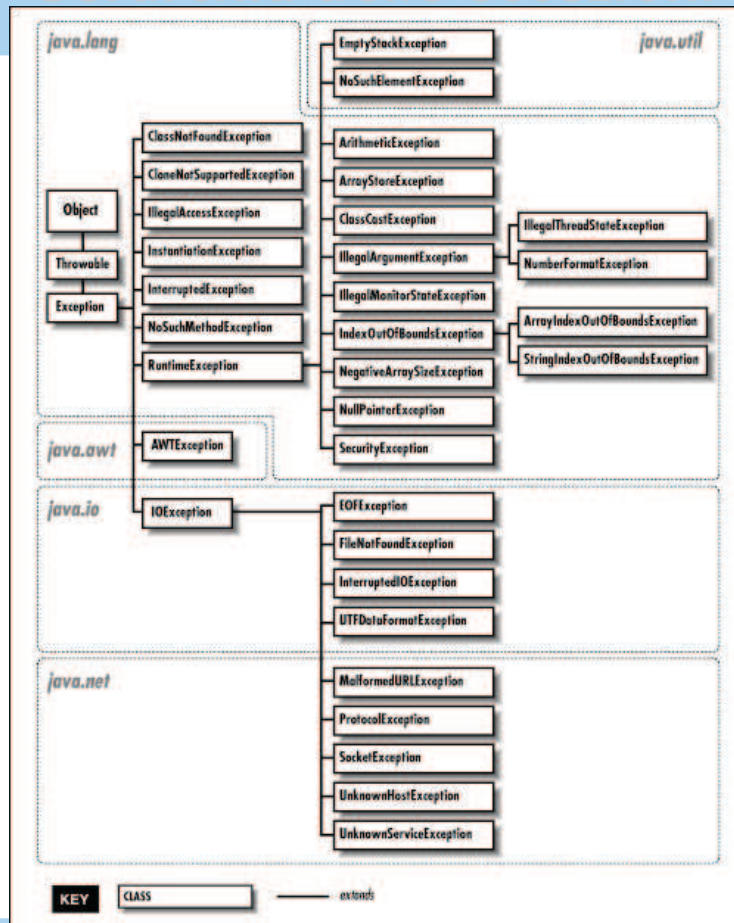
STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Gerarchia delle eccezioni



STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Gerarchia delle eccezioni



STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Eccezioni come oggetti - 1

Riprendiamo il nostro esempio: quando si verifica un'eccezione nel metodo `parseInt` viene creata un'istanza di una sottoclasse di `Exception` (in questo caso `NumberFormatException`)

Questa istanza viene passata al blocco `catch`

```
catch (Exception ex)
{
    a = 0;
}
```

La variabile `ex` (il nome non è fisso) è quindi un riferimento all'istanza di `NumberFormatException`

Essendo di tipo `Exception`, in virtù del `subtyping`, la variabile `ex` può puntare ad istanze di una qualunque sottoclasse di `Exception`

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Eccezioni e metodi

- ✓ Dal momento che un'eccezione è un oggetto possiamo invocare su di essa i metodi definiti dalla classe a cui appartiene.

In particolare tutte le eccezioni implementano il metodo `getMessage()` (è definito nella classe base `Throwable`)

`getMessage()` fornisce una descrizione dell'eccezione

Potremmo quindi scrivere:

```
catch (Exception ex)
{
    System.out.println(ex.getMessage());
}
```

Le sottoclassi possono poi definire metodi specifici che forniscono ulteriori informazioni.

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Una gestione più accurata

La gerarchia delle eccezioni e la possibilità di avere più blocchi `catch` consente di differenziare la gestione delle eccezioni.

```
try
{
    a = Integer.parseInt(s);
}
catch (NumberFormatException e)
{
    a = 0;
}
catch (Exception e)
{
    System.out.println(e.getMessage()); System.exit(1);
}
```

- ✓ In questo modo gestiamo in maniera completa l'eccezione specifica che ci interessa e in maniera generica le altre

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE



## Catch multipli

I blocchi catch vengono gestiti in cascata: se un'eccezione non è del tipo specificato si passa a quello successivo

Mettendo in fondo un blocco che ha Exception come tipo si catturano tutte le eccezioni (però non gli errori di sistema )

Quindi:

- Se si verifica un'eccezione di tipo **NumberFormatException** viene eseguito il primo blocco che recupera la situazione attribuendo un valore di default ad **a**
- In tutti gli altri casi di eccezione viene eseguito il secondo blocco catch: si mostra a video un messaggio e si esce dal programma

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## riassumendo...

```
try {  
  
    //operazione critica che può sollevare eccezioni  
}  
catch (Exception1 e1) {  
    //gestione dell'eccezione  
}  
catch (Exception2 e2) {  
    //gestione dell'eccezione  
}  
finally {  
    //codice da eseguire comunque dopo il blocco try  
}
```

Se l'operazione solleva **diversi tipi di eccezione in risposta a diversi tipi di errore**, più blocchi **catch** possono seguire lo stesso blocco **try**

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Rilanciare le eccezioni

Java prevede un meccanismo per garantire una gestione corretta delle eccezioni.

Un metodo in cui si può verificare un'eccezione è obbligato a fare una delle seguenti due cose:

- **Gestire l'eccezione**, con un costrutto try/catch
- **Rilanciarla** esplicitamente all'esterno del metodo, delegandone in pratica la gestione ad altri

Se si sceglie questa seconda strada, il metodo deve indicare quale eccezione può "uscire" da esso, con la clausola **throws**.

☒ **Se non lo fa, il compilatore dà un errore**

Si crea quindi una **catena di reponsabilità** nella gestione delle situazioni critiche: ad ogni livello possiamo quindi decidere se l'azione correttiva può essere eseguita o se dobbiamo rimandarla più in alto

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Gestione o rilancio

Nel nostro esempio potremmo quindi agire in due modi:

### Gestione

```
public class EsempioEcc2
{
    public static void
        main(String args[])
    {
        int a = 0;
        String s = args[0];
        try
        {
            a = Integer.parseInt(s);
        }
        catch (Exception e)
        { a = 0; }
    }
}
```

### Rilancio

```
public class EsempioEcc1
{
    public static void
        main(String args[])
        throws
            NumberFormatException
    {
        int a = 0;
        String s = args[0];
        a = Integer.parseInt(s);
    }
}
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Lanciare eccezioni - 1

Anche nei metodi scritti da noi possiamo generare eccezioni per segnalare situazioni anomale

Definiamo per esempio una classe che consente di convertire stringhe in numeri solo per numeri <1000

```
public class Thousand
{
    public static int parseInt(String s)
        throws NumberFormatException
    {
        int a = Integer.parseInt(s);
        if (a >= 1000)
        {
            NumberFormatException e = new NumberFormatException();
            throw e;
        }
    }
}
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Lanciare eccezioni - 2

Quindi:

1. Prima **si crea l'oggetto eccezione** da lanciare, come istanza di una sottoclasse di Exception
2. Poi lo si lancia con l'istruzione **throw**

Il metodo deve inoltre dichiarare che **può mandare all'esterno un'eccezione `NumberFormatException`**, che può essere generata da **`Integer.parseInt()`** oppure dal metodo stesso

💣 **Attenzione:** non bisogna confondere la clausola **throws** con l'istruzione **throw**:

- ✓ **throw** genera (si dice anche **solleva**) un'eccezione
- ✓ **throws** dichiara che un metodo rilancia all'esterno un'eccezione

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# Definizione di eccezioni

Nell'esempio precedente abbiamo utilizzato un tipo di eccezione predefinito (**NumberFormatException**)

Possiamo però definire un'eccezione specifica per il nostro scopo.

Per far questo è sufficiente definire una sottoclasse di Exception:

```
public class NumberTooBigException extends Exception
{
    public NumberTooBigException() { super(); }
    public NumberTooBigException(String s) { super(s); }
}
```

Dobbiamo **definire i due costruttori** standard:

- Quello di default
- Quello con un parametro stringa (il messaggio)

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## Esempio con eccezione definita

Il nostro esempio diventa quindi:

```
public class Thousand
{
    public static int parseInt(String s)
        throws NumberFormatException,
               NumberTooBigException
    {
        int a = Integer.parseInt(s);
        if (a >= 1000)
        {
            NumberTooBigException
                e = new NumberTooBigException();
            throw e;
        }
    }
}
```

Dobbiamo dichiarare che il metodo può emettere due tipi di eccezioni: quella di **Integer.parseInt()** e la nostra

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## La clausola finally

L'istruzione try prevede una clausola finally opzionale:

```
try
{...}
catch (Exception e)
{...}
finally
{...}
```

Il blocco finally deve essere messo sempre alla fine

Le istruzioni del blocco finally vengono eseguite comunque:

- In assenza di eccezioni il blocco finally viene eseguito subito dopo il blocco try
- Se si verificano eccezioni viene eseguito prima l'eventuale blocco catch e poi il blocco finally

E' possibile utilizzare finally senza che siano presenti blocchi catch.

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

## le eccezioni - esercizio

Creare una classe per la conversione stringa>numero intero.

La conversione deve essere limitata all'intervallo [-10, 10], nel caso il numero sia fuori dall'intervallo deve essere lanciata l'eccezione **IllegalArgumentException**.

Il costruttore della classe deve lanciare tutte le eventuali eccezioni che devono essere catturate dal **main**.

Il parametro da convertire deve essere dato in ingresso al **main**.

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# le eccezioni - soluzione

Creo la classe e il costruttore.

```
public class Esercizio1 {

    /** Il costruttore. */
    public Esercizio1(String[] args) throws
        NumberFormatException,
        IllegalArgumentException {
        String numero = args[0];
        int num = Integer.parseInt(numero);
        if((num<-10) || (num>10)) {
            IllegalArgumentException e =
                new IllegalArgumentException("Fuori dall'intervallo");
            throw e;
        }
        System.out.println("Numero: " + num);
    }
}
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE

# le eccezioni - soluzione

Creo il main (all'interno della classe).

```
... ..
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {

    try {
        Esercizio1 oggetto = new Esercizio1(args);
    }
    catch(NumberFormatException e1) {
        System.out.println("Format Exception 1: "+e1.getMessage());
    }
    catch(IllegalArgumentException e2) {
        System.out.println("Format Exception 2: "+e2.getMessage());
    }

}

} //fine della classe
```

STRUMENTI JAVA PER LO SVILUPPO DI INTERFACCE UTENTE E SERVIZI DI RETE E LORO APPLICAZIONE