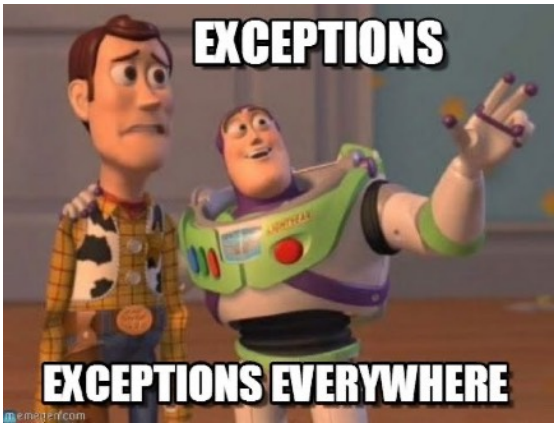
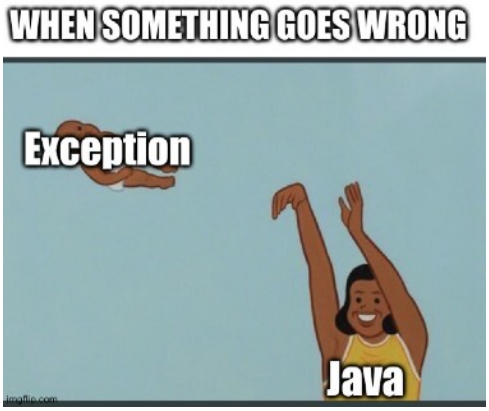
Documento a cura di Simone Remoli.

# LE ECCEZIONI IN JAVA.

**"Problem Exists Between Keyboard And Chair"**

Le eccezioni sono fondamentali perché un programma non può prevedere **tutte** le possibili situazioni anomale che possono verificarsi durante l'esecuzione.
Se un programma non gestisse le eccezioni, ogni minimo errore potrebbe mandarlo in crash. Quindi, più che per proteggere il programma dall'utente "scemo", le eccezioni servono a rendere il codice **più robusto e affidabile**.




Facciamo una panoramica.

```
 try {
    System.out.println("Inserire il numero di ciambelle: ");
    int conteggioCiambelle = tastiera.nextInt();

    System.out.println("Inserire il numero di bicchieri di latte: ");
    int conteggioLatte = tastiera.nextInt();

    if (conteggioLatte < 1) {
        throw new Exception("Eccezione: Niente latte!");
    }
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("Vai a comprare il latte :) ");
}
```

Out:

```
Inserire il numero di ciambelle:
1
Inserire il numero di bicchieri di latte:
0
java.lang.Exception Create breakpoint : Eccezione: Niente latte!
    at mio.pacchetto.prova.Main.main(Main.java:22)
Vai a comprare il latte :)
```

In questo esempio viene generata un'**eccezione generale** perché si usa direttamente la classe `Exception`, che è la classe base per tutte le eccezioni controllate (*checked exceptions*) in Java.

Come si lanciano le eccezioni? **throw new** nome_classe_eccezione("frase");

`Exception` è la classe madre di tutte le eccezioni "**gestibili**" in Java, ma la vera classe madre di tutte le eccezioni è `Throwable`.

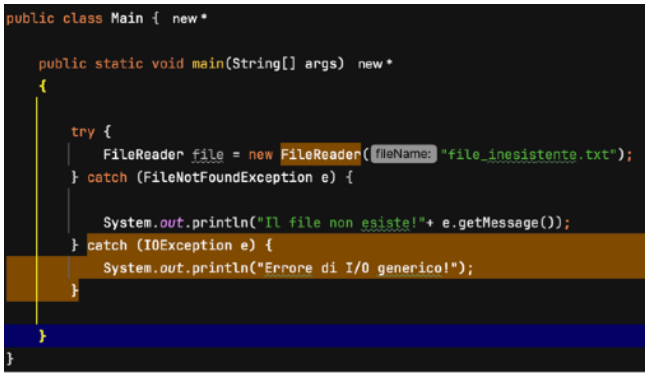
Ora il manuale è di fondamentale importanza:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Exception.html
Al link sono elencate tutte le eccezioni.

**Adesso vedrete una tabella che può aiutare la comprensione delle eccezioni.**
**Se nel tuo codice catturi `IOException`, essa verrà sollevata se si verifica una qualsiasi delle eccezioni derivate elencate nella tabella sulla destra.**

`IOException è la superclasse di eccezioni come FileNotFoundException, SocketException, EOFException, ecc.`
`Questo significa che se una di queste sottoclassi viene sollevata, può essere catturata con un catch generale su IOException.`

```
public static void main(String[] args)
{
        try
        { // Tenta di aprire un file che non esiste
                FileReader file = new FileReader("file_inesistente.txt");
         } catch
         (IOException e)
         {
                System.out.println("Errore di I/O: " + e.getMessage());
         }
}
```

**Risultato: Questo codice catturerà anche `FileNotFoundException`, perché essa è una sottoclasse di `IOException`.**
**Codice con gestione più dettagliata:**




Qui **distinguo tra errori diversi**, così posso dare messaggi più specifici. Se usi `IOException`, essa intercetterà qualsiasi delle sue sottoclassi.
**Se vuoi una gestione più precisa, intercetta direttamente le sottoclassi di `IOException`.**

# THROW.

In Java, la parola chiave **throw** viene utilizzata per generare manualmente un'eccezione durante l'esecuzione del programma. Quando scrivi **throw new Exception("Errore!")**, stai dicendo alla JVM di interrompere il flusso normale del programma e di cercare un blocco **catch** che possa gestire l'eccezione.

## Differenza tra throw e throws.

| Parola Chiave | Significato | Dove si usa? |
|---|---|---|
| `throw` | Solleva un'eccezione specifica | Dentro un metodo |
| `throws` | Dichiarazione che un metodo **potrebbe** lanciare un'eccezione | Nella firma del metodo |

Ecco un semplice esempio di utilizzo di **throw** per generare manualmente un'eccezione:

```
public class TestThrow {
    public static void main(String[] args) {
        try {
            verificaNumero(-5);
        } catch (IllegalArgumentException e) {
            System.out.println("Eccezione catturata: " + e.getMessage());
        }
    }

    public static void verificaNumero(int num) {
        if (num < 0) {
            throw new IllegalArgumentException("Numero negativo non ammesso!");
        }
        System.out.println("Numero valido: " + num);
    }
}
```

Il metodo verificaNumero(-5) lancerà un'IllegalArgumentException, che verrà catturata nel blocco catch.

Di seguito una tabella delle eccezioni comuni:

# Throwable

| | | | |
|---|---|---|---|
| Exception | AbsentInformationException | | Viene lanciata quando le informazioni di debug non sono disponibili. |
| | AgentInitializationException | | Viene lanciata quando un Java Agent non riesce a inizializzarsi correttamente nella JVM di destinazione. |
| | AgentLoadException | | Viene lanciata quando un Java Agent non può essere caricato nella JVM di destinazione. |
| | **IOException** (Si verifica se almeno una delle eccezioni a destra avviene) | FileNotFoundException | Il file richiesto non esiste o il percorso è errato. |
| | | FileSystemException | Errore generico legato al filesystem (permessi, spazio insufficiente, ecc.). |
| | | FileLockInterruptionException | Un thread è stato interrotto mentre aspettava un file lock |
| | | InvalidPropertiesFormatException | Il file .properties ha un formato non valido |
| | | SyncFailedException | Il sistema non riesce a sincronizzare un file sul disco. |
| | | ZipException | Errore durante la lettura o scrittura di un file ZIP (archivio corrotto, formato errato). |
| | | SocketException | Errore generico sui socket (connessione chiusa, timeout, porta occupata). |
| | | SSLException | Problemi di sicurezza nella connessione SSL/TLS (certificato scaduto, handshake fallito). |
| | | WebSocketHandshakeException | Errore nella negoziazione della connessione WebSocket. |
| | | EOFException | Il flusso di dati è terminato inaspettatamente (End Of File). |
| | **RuntimeException** | IllegalArgumentException | Passaggio di un argomento non valido a un metodo. |
| | | IndexOutOfBoundsException | Tentativo di accedere a un indice fuori dai limiti di un array o lista. |
| | | ArithmeticException | Errore nelle operazioni matematiche (es. divisione per zero). |
| | | NegativeArraySizeException | Creazione di un array con dimensione negativa. |
| | | ClassCastException | Tentativo di eseguire un casting non valido tra tipi di oggetti. |
| | **SQLException (Un'eccezione che fornisce informazioni su un errore di accesso al database o altri errori.)** | SQLNonTransientException | Errore SQL permanente, ovvero non risolvibile riprovando la stessa operazione. Tipici errori: sintassi SQL errata, violazioni di vincoli. |
| | | SQLTransientException | Errore SQL temporaneo, che potrebbe risolversi da solo riprovando l'operazione. Tipici casi: deadlock, blocchi temporanei nel database. |

Un esempio:

```java
public class Main {  new *
    private static void ciao(int a) throws Exception  1 usage  new *
    {
        if(a<0)
            throw new Exception("Errore nell'inserimento argomento");
        a = a + 1;
    }
    public static void main(String[] args)  new *
    {
        int l = -10;
        try {
           ciao(l);
        } catch (Exception e)
        {
            e.printStackTrace();
            System.out.println("Errore argomento."+ e.getMessage());

        }
    }

}
```

Out:

```
java.lang.Exception Create breakpoint : Errore nell'inserimento argomento
    at mio.pacchetto.prova.Main.ciao(Main.java:14)
    at mio.pacchetto.prova.Main.main(Main.java:23)
Errore argomento.Errore nell'inserimento argomento
```

Se levassi la stampa dello stack delle chiamate allora si leverebbe anche il rosso.
Il metodo ciao può lanciare delle eccezioni di tipo Exception, appena le lancia c'è un try catch che ingloba il metodo ciao e gestisce la medesima eccezione nel catch.

Il metodo può generare anche eccezioni specifiche:

```java
public class Main {  new *
    private static void ciao(int a) throws IllegalArgumentException  1 usage  new *
    {
        if(a<0)
            throw new IllegalArgumentException("Errore nell'inserimento argomento");
        a = a + 1;
    }
    public static void main(String[] args)  new *
    {
        int l = -10;
        try {
           ciao(l);
        } catch (IllegalArgumentException e)
        {
            e.printStackTrace();
            System.out.println("Errore argomento."+ e.getMessage());

        }
    }

}
```

Out:

```
java.lang.IllegalArgumentException Create breakpoint : Errore nell'inserimento argomento
    at mio.pacchetto.prova.Main.ciao(Main.java:7)
    at mio.pacchetto.prova.Main.main(Main.java:14)
Errore argomento.Errore nell'inserimento argomento
```

Non è detto che un metodo throws generi un’eccezione esplicita con throw.

```java
public class Main {  new *
    private static void ciao(int a) throws IOException  1 usage  new *
    {
        Scanner tastiera = new Scanner(System.in);
        int l = tastiera.nextInt();
        System.out.println("dopo");
    }
    public static void main(String[] args)  new *
    {
        int l = -10;
        try {
           ciao(l);
        } catch (IOException e)
        {
            e.printStackTrace();
            System.out.println("Hai inserito una stringa ad un intero."+ e.getMessage());

        }
    }
}
```

Out:

```
ER
Exception in thread "main" java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:947)
    at java.base/java.util.Scanner.next(Scanner.java:1602)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2267)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2221)
    at mio.pacchetto.prova.Main.ciao(Main.java:10)
    at mio.pacchetto.prova.Main.main(Main.java:17)
```

Oppure:

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class Main {  new *
    private static void ciao(int a) throws InputMismatchException  1 usage  new *
    {
        Scanner tastiera = new Scanner(System.in);
        int l = tastiera.nextInt();
        System.out.println("dopo");
    }
    public static void main(String[] args)  new *
    {
        int l = -10;
        try {
           ciao(l);
        } catch (InputMismatchException e)
        {
            //e.printStackTrace();
            System.out.println("Hai inserito una stringa ad un intero."+ e.getMessage());

        }
    }
}
```

Out:

```
00
Hai inserito una stringa ad un intero.null

Process finished with exit code 0
```

Se decremento lo stack delle chiamate:

```
oo
java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:947)
    at java.base/java.util.Scanner.next(Scanner.java:1602)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2267)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2221)
    at mio.pacchetto.prova.Main.ciao(Main.java:10)
    at mio.pacchetto.prova.Main.main(Main.java:17)
Hai inserito una stringa ad un intero.null
```

Non c'è throw però.

---

Ora qualsiasi eccezione (inclusa **InputMismatchException**) verrà catturata dal **catch (Exception e).**

```java
public class Main {  new *
    private static void ciao(int a) throws Exception {  // Generale: può lanciare qualsiasi eccezione  1 usage  new *
        Scanner tastiera = new Scanner(System.in);

        int l = tastiera.nextInt();  // Se l'input non è un numero, genera InputMismatchException
        System.out.println("dopo");
    }

    public static void main(String[] args) {  new *
        int l = -10;

        try {
            ciao(l);
        } catch (Exception e) {  // Cattura qualsiasi eccezione, inclusa InputMismatchException
            e.printStackTrace();
            System.out.println("Errore generico catturato: " + e.getMessage());
        }
    }
}
```

Out:

```
oo
java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:947)
    at java.base/java.util.Scanner.next(Scanner.java:1602)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2267)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2221)
    at mio.pacchetto.prova.Main.ciao(Main.java:9)
    at mio.pacchetto.prova.Main.main(Main.java:17)
Errore generico catturato: null
```