

## Eccettive, Definizione e meccanismo

Sono costrutti che alterano il normale flusso del codice, inserendo **eventi eccezionali**

Il linguaggio supporta i seguenti meccanismi:

- Throw → **lanciare**
- Throws → metodi che lanciano un'eccezione
- Try-catch → blocco dedicato alla cattura di un'eccezione

**Lanciare** → le eccezioni sono oggetti e ognuna è un'istanza delle sottoclassi di **Throwable**

Un'eccezione viene lanciata con **Throw <exp>;**  
di tipo dichiarato **Throwable**  
o **sottotipo**

Esempio:

**Throw new IllegalStateException();**

bloco di vita di un'eccezione

Un lancio interrompe il normale flusso di esecuzione:

- 1 Lanciate cattura locale → l'eccezione termina il metodo corrente e passa al chiamante, con la possibilità di catturare;
- 2 Lanciate cattura del metodo chiamante → l'eccezione risale lo stack di attivazione, fino al main;
- 3 Lanciate cattura anche dal main → Il programma termina e la JVR stampa il contenuto dell'eccezione (**stack trace**)

Quando un'eccezione viene creata, essa registra la situazione corrente dello stack  
Esempio:

```
1 public class MyClass {  
2     public static void main(String[] args) {  
3         (new MyClass()).foo();  
4     }  
5     public void foo() {  
6         bar();  
7     }  
8     public void bar() {  
9         throw new RuntimeException();  
10    }  
11}
```

Output:

**Thread principale**

Exception in thread "main" java.lang.RuntimeException

at MyClass.bar(MyClass.java:11)

at MyClass.foo(MyClass.java:7)

at MyClass.main(MyClass.java:3)

sottoclasse di **Throwable**

dovrebbene avere le righe, ma l'immagine è sbagliata

## Cattura delle eccezioni

La parte di codice che potrebbe lanciare eccezioni viene inserito nel blocco **try**, al cui interno sono poste le clausole **catch** per gestire le eccezioni.

```
try {  
    // Codice "rischioso"  
} catch (Eccezione1 e) {  
    // Codice che gestisce una Eccezione1  
} catch (Eccezione2 e) {  
    // Codice che gestisce una Eccezione2  
}  
// Codice non rischioso
```

"classi qualsiasi"

## Blocco finally

Blocco secondario che verrà sempre eseguito, anche dopo la gestione

```
try {  
    // Codice "rischioso"  
} catch (Eccezione1 e) {  
    // Codice che gestisce una Eccezione1  
} catch (Eccezione2 e) {  
    // Codice che gestisce una Eccezione2  
} finally {  
    // Codice da eseguire in ogni caso  
}  
// Codice non rischioso
```

Anche dopo un ~~return~~ altro try/catch

Finally solo grazie ad un crash potrebbe fallire

## Sequenza di esecuzione

Supponiamo che nei vari blocchi non ci siano istruzioni return:

```
try {  
    // (1) Codice "rischioso"  
} catch (Eccezione1 e) {  
    // (2) Codice che gestisce una Eccezione1  
} catch (Eccezione2 e) {  
    // (3) Codice che gestisce una Eccezione2  
} finally {  
    // (4) Codice da eseguire in ogni caso  
}  
// (5) Codice non rischioso
```

- Sequenza se non viene lanciata alcuna eccezione: (1) (4) (5)
- Sequenza se viene lanciata Eccezione1: (1 fino al lancio) (2) (4) (5)

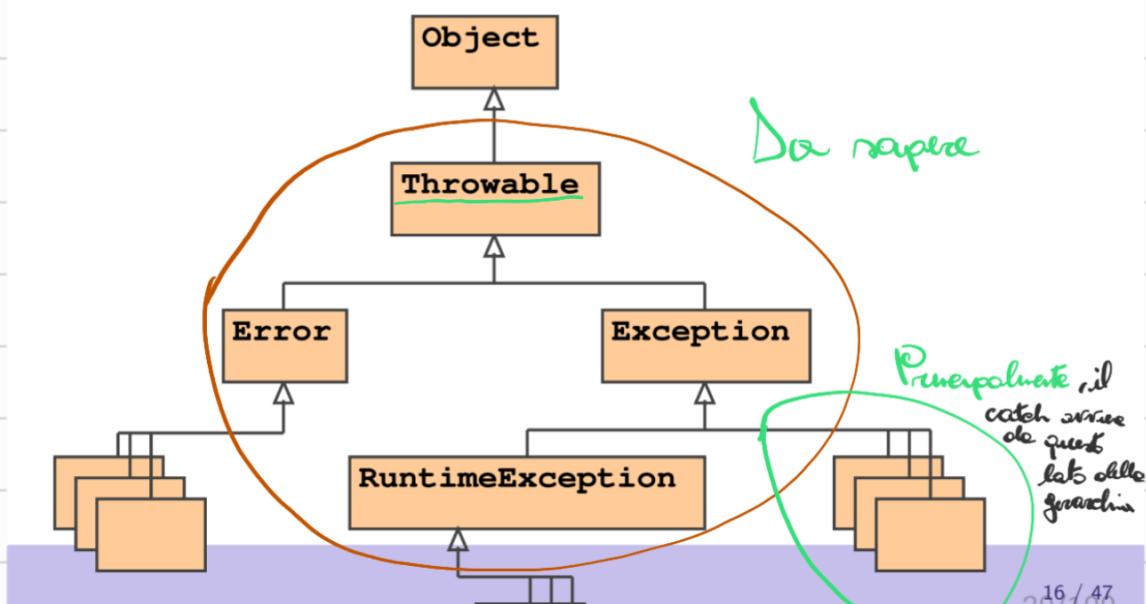
Se manca il catch adatto, la sequenza sarebbe 1 - 9

# Vineoli sentattici

- Catch e finally non si sovrapposono, ma try deve contenere almeno uno dei due
- Un try solitario causa un errore di compilazione
- Se esistono diverse clausole catch, devono immediatamente seguirsi try
- Finally deve stare per ultimo
- L'ordine delle clausole è importante

## Generichità

Ecco le principali classi di eccezioni:



**Throwable** rappresenta tutti gli oggetti lanciabili e contiene `printStackTrace();`

Le classi **Error** e le sue sottoclassi coprono gli errori non causati dal programmatore, e spesso queste eccezioni non vengono catturate.

La classe **RuntimeException** copre gli errori di programmazione e pertanto non vengono catturate.

## Eccezioni catturate

**catch (E e)** cattura ogni oggetto-**eccezione** il cui tipo effettivo è **sottotipo di E**

↳ verrà catturata dal primo blocco try  
capace di gestirla

Esempio: la classe `IndexOutOfBoundsException` ha due sottoclassi, `ArrayIndexOutOfBoundsException` e `StringIndexOutOfBoundsException`; si può scrivere una unica clausola che catturi una qualunque di queste eccezioni:

```
try {  
    // Codice che potrebbe lanciare una eccezione  
    // IndexOutOfBoundsException oppure  
    // ArrayIndexOutOfBoundsException oppure  
    // StringIndexOutOfBoundsException  
}  
catch (IndexOutOfBoundsException e) {  
    e.printStackTrace();  
}
```

**Consiglio** → Scrive le catch più specifiche in alto per evitare errori.

Casi particolari → Le eccezioni lanciate dentro i blocchi catch e finally non vengono gestite da altri catch dello stesso try (valido uno try-catch annidati)

### Eccezioni catturate (2)

- Resistere alla tentazione di scrivere una unica clausola catch-all:

```
try {  
    // codice rischioso  
} catch (Exception e) {  
}
```

- L'ordine delle clausole catch è importante.
- Nell'esempio precedente, se avessimo scritto:

```
try {  
    ...  
} catch (IndexOutOfBoundsException e) { Troppo generale  
    ...  
} catch (ArrayIndexOutOfBoundsException e) { // Err. di comp. Specifico  
    ...  
}
```

il codice non sarebbe stato compilato, perché il secondo catch è ridondante

Per evitare problemi basta invertire le clausole catch

## Categorie di eccezioni

Possono essere checked e unchecked

Tutte le altre ↴ ↴ eccezioni come Error e RuntimeException e sottoclassi  
Varie

soggette a handle - or - declare

## Clausola throws

Per specificare quali eccezioni un metodo può lanciare, usiamo throws

```
void miaFunzione() throws MiaEccezione1, MiaEccezione2 {  
    ...  
}
```

## Handle or Declare

Se un metodo lancia un'eccezione checked, o richiama un altro metodo che può farlo, abbiamo due opzioni ( regola handle - or - declare )

1 Cattura e gestisci

2 Dichiarare l'eccezione nell'intestazione del metodo

]} "Verificata" indica che queste regole

# Esempi

Quali problemi ci sono in questo codice?

```
void f1() {  
    f2();  
}  
  
void f2() {  
    throw new IOException();  
}
```

Versione corretta  
Void f2() throws IOException { ... }

- Il metodo f2 lancia una eccezione checked ma non la dichiara; questo è un errore di compilazione

Anche se il problema si sposta a f1 che dovrebbe gestire

È possibile creare anche delle proprie eccezioni, estendendo una qualsiasi eccezione esistente

Class MyException extends Exception { ... }

## Overriding

Punto che sono sovrsovrribili solo i metodi visibili della superclasse, ecc le regole che completano l'override:

- 3 metodi devono avere la stessa firma
- Il tipo di ritorno può essere uguale o sottotipo del tipo di ritorno di partenza
- Nessun dei due metodi può essere static

- Il metodo della superclasse non più essere find
- " " " " sottoclasse non oltre avere visibilità superiore
- Se il metodo della sottoclasse dichiara di lanciare un'eccezione **checked**, tale tipo deve essere sottolista rispetto al metodo della super.

La gerarchia deve essere rispettata

