Eccezioni predefinite

- Java mette a disposizione molte eccezioni già confezionate, che descrivono la maggioranza dei problemi che possono verificarsi a run-time.
- Ognuna di queste è una classe.
  - Tutte queste classi estendono, direttamente o indirettamente, la classe **`Exception`** definita nel package **`java.lang`** (come discuteremo, estendono tutte anche **`RuntimeException`**)
  - Tra di loro, possono avere ulteriori relazioni di gerarchia ereditaria.

Eccezioni predefinite

- Esempi notevoli

`ArithmeticException` : viene sollevata per problemi di calcolo (es. divisione per zero)

`ArrayIndexOutOfBoundsException` : viene sollevata se si cerca di accedere a una posizione inesistente in un array

`ClassCastException` : viene sollevata se si tenta di fare un cast non lecito: l'oggetto che si vuole "castare" non è istanza di quella classe

Eccezioni predefinite

- Esempi notevoli (continua)

`NullPointerException` (molto frequente) : si tenta di accedere all'oggetto riferito da una variabile, ma il riferimento è a **`null`**

`NumberFormatException` : si tenta di convertire in un numero qualcosa senza il giusto formato

`StringIndexOutOfBoundsException` : viene sollevata se si cerca di accedere a una posizione inesistente in una stringa

Sollevare eccezioni

- Oltre a contrastare le eccezioni, potremmo anche volerle crearle intenzionalmente.
- A tale scopo esiste il comando **`throw`** (=lancia)
- Sintassi:

```
throw istanza_oggetto_eccezione;
```

  dove **`istanza_oggetto_eccezione`** è un'istanza della classe **`Throwable`**.

  - Deve essere **un'istanza** (creata ad esempio con **`new`**) e non solo il nome di una classe!

Sollevare eccezioni

- Quella che lanciamo intenzionalmente è un'eccezione vera e propria, quindi si propaga a ritroso tra i metodi; inoltre invocare **`throw`** **termina** l'esecuzione del metodo corrente
  - È come usare **`return`**, ma in questo caso si tratta di terminazione anomala. Non c'è quindi valore di ritorno, semmai si "ritorna" l'eccezione.
- Il controllo ritorna così al metodo chiamante.
  - Quest'ultimo non può riesumare il metodo che ha lanciato l'eccezione: può solo reinvocarlo.

Sollevare eccezioni

- Consideriamo ancora l'esempio di **`Divisione`**, il cui **`main()`** invoca **`calcolaQuoto(a,b)`**.
  - Potremmo decidere di lanciare un'eccezione dentro al metodo **`calcolaQuoto()`** se **`b==0`**
  - Per fare ciò bisogna **prima** costruire un oggetto di tipo **`ArithmeticException`**, e **poi** lanciarlo.
- Per costruire l'eccezione si usa un costruttore:

```
new ArithmeticException("Messaggio");
```

  - ha un parametro **`String`** (msg d'errore associato)

Sollevare eccezioni

- L'eccezione va lanciata scrivendo **`throw`** e un riferimento a oggetto eccezione. Quindi:

```
throw new ArithmeticException("Errore!");
```

  e l'eccezione è anonima. Ma va bene anche:

```
ArithmeticException e;
e = new ArithmeticException("Errore!");
throw e;
```

- Nota: anche se l'eccezione è riferita da una variabile locale, quest'ultima viene distrutta subito perché l'eccezione termina il metodo.

Sollevare eccezioni

- Quindi non importa che nome si dà (se lo si dà) all'eccezione nel metodo che la crea.
  - Invece, importa come chiamarla nel corpo della catch del metodo che la gestisce.
- Va usato l'identificativo dato all'atto della cattura

```
} catch ( Exception ecc )
{ ecc.metodo(); // non e.metodo()!!
```

  - In effetti, invece di "**ecc**" la si poteva chiamare pure come nel metodo che l'ha creata (in questo caso, **e**) , tanto quel nome è stato distrutto e non esiste più (nessun rischio di shadowing).

Sollevare eccezioni

- Le classi di eccezioni sono sottoclassi della classe **`Throwable`**, che ha due costruttori:
  - con un parametro: **`Throwable(String msg)`** dove **`msg`** è il messaggio d'errore.
  - senza parametri: **`Throwable()`**
- Tramite opportune invocazioni del costruttore **`super()`**, questi costruttori possono essere propagati lungo la gerarchia d'ereditarietà.
  - Questo è stato fatto per tutte le eccezioni predefinite: ed è una pratica da rispettare!

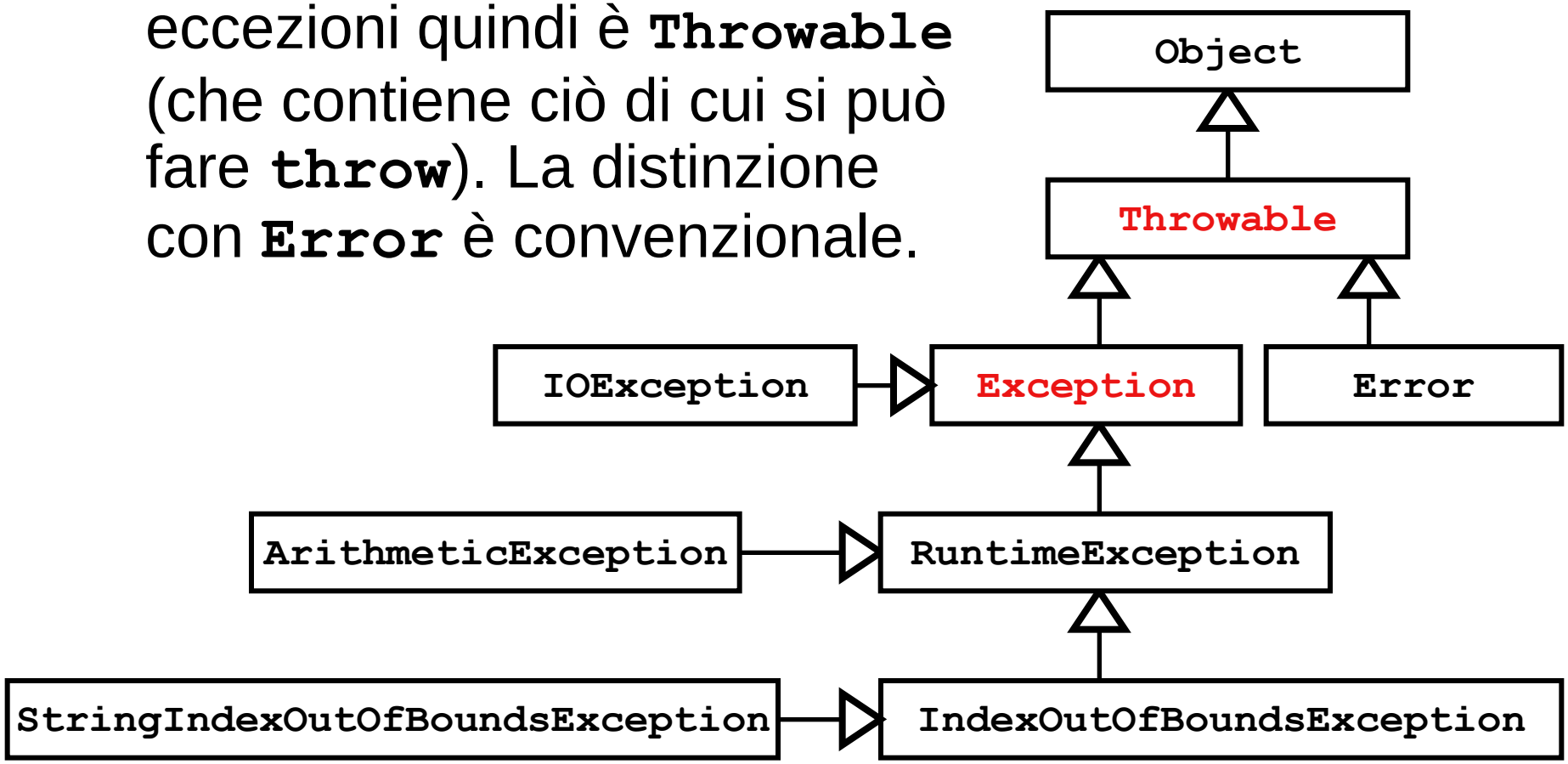
Sollevare eccezioni

- Tutte le eccezioni hanno poi i seguenti metodi, che vengono ereditati dalla classe **`Throwable`**:
  - **`toString()`** : ritorna il nome della classe e il messaggio con cui l'oggetto è stato costruito.
  - **`getMessage()`** : ritorna solo il messaggio.
  - **`printStackTrace()`** : stampa il punto dove è stata sollevata l'eccezione

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

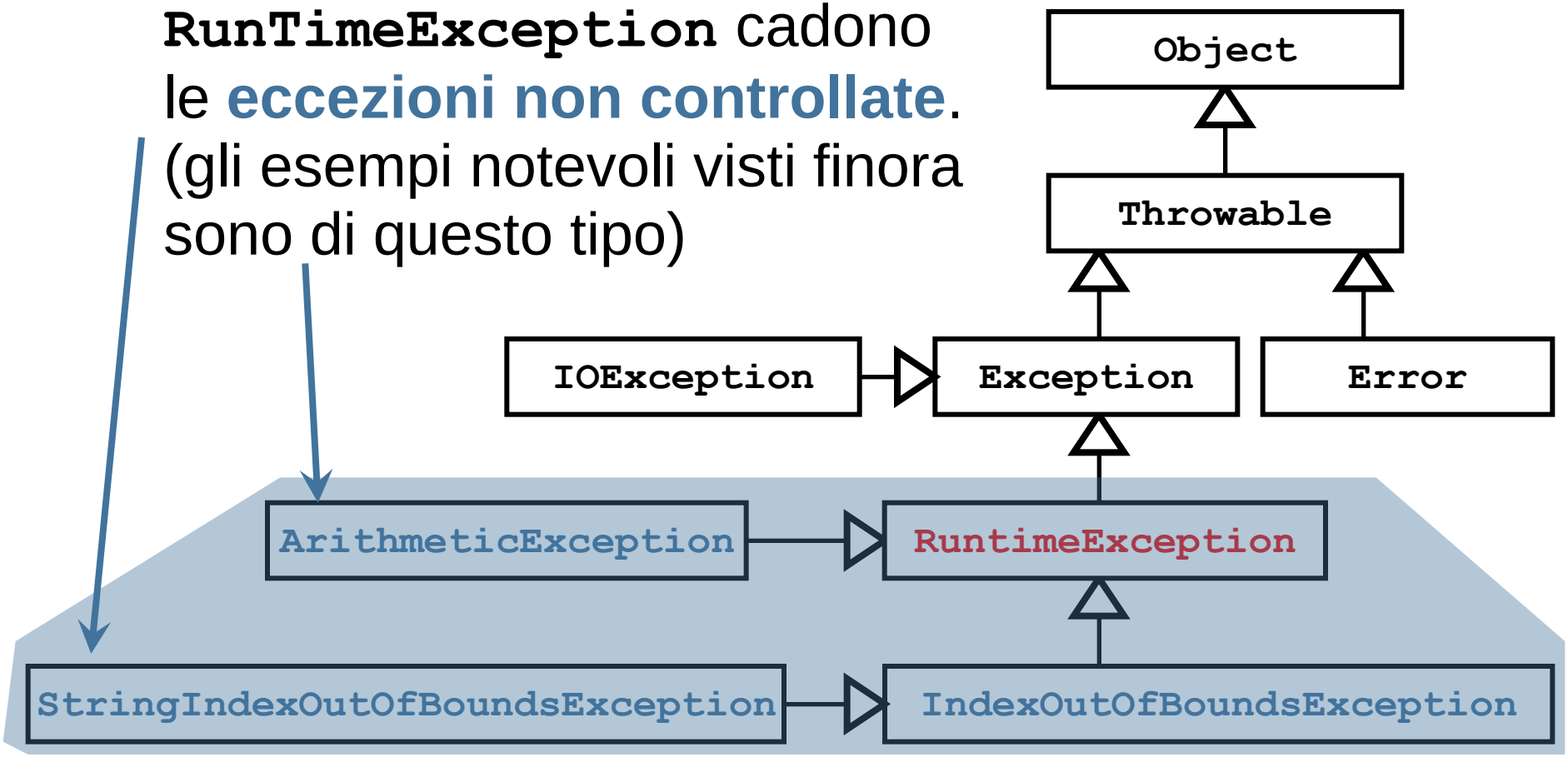
Sollevare eccezioni

- In realtà la superclasse delle eccezioni quindi è **`Throwable`** (che contiene ciò di cui si può fare **`throw`**). La distinzione con **`Error`** è convenzionale.



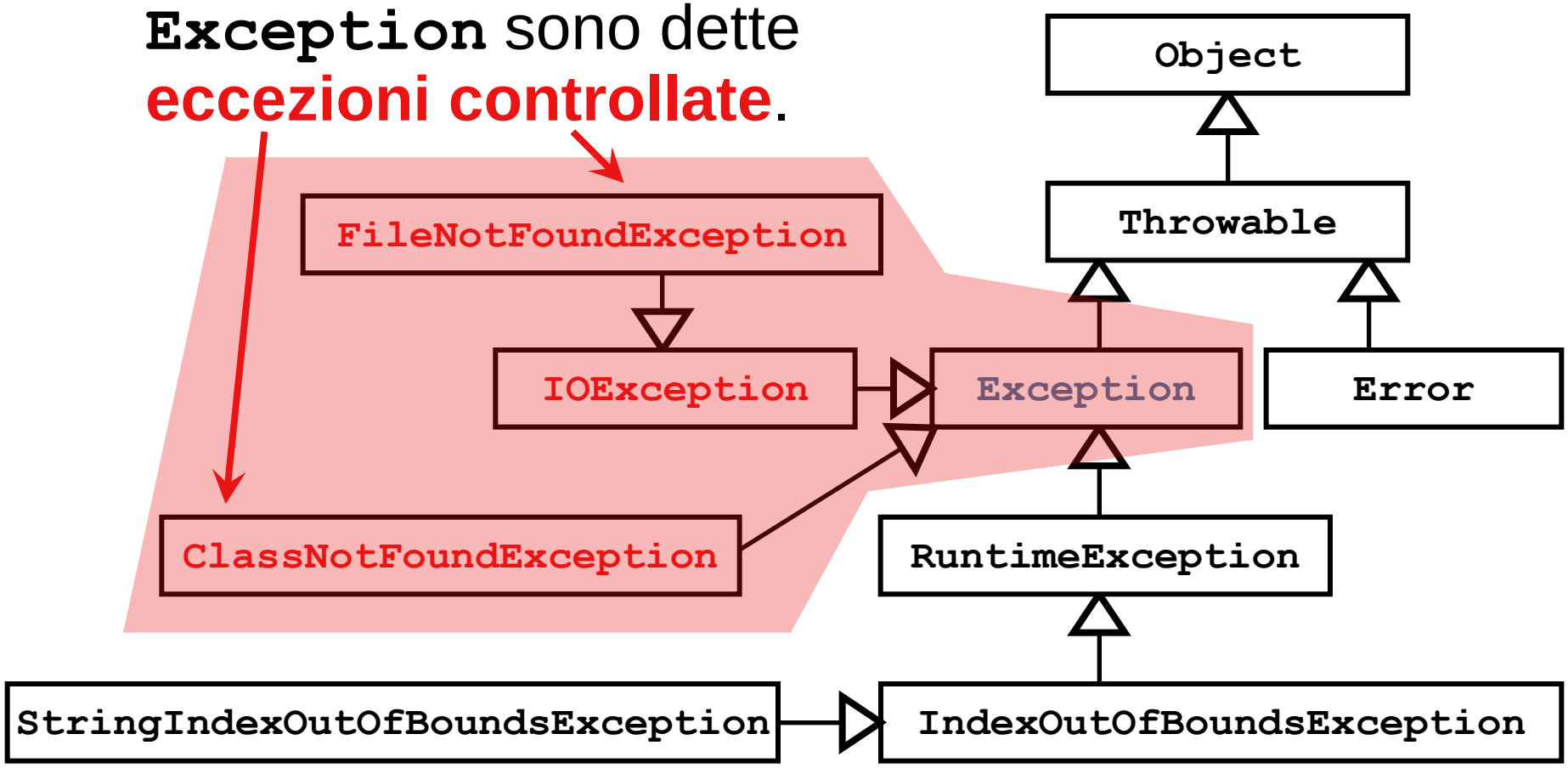
Eccezioni non controllate

- Sotto l'ulteriore sottoclasse **`RunTimeException`** cadono le **eccezioni non controllate**. (gli esempi notevoli visti finora sono di questo tipo)



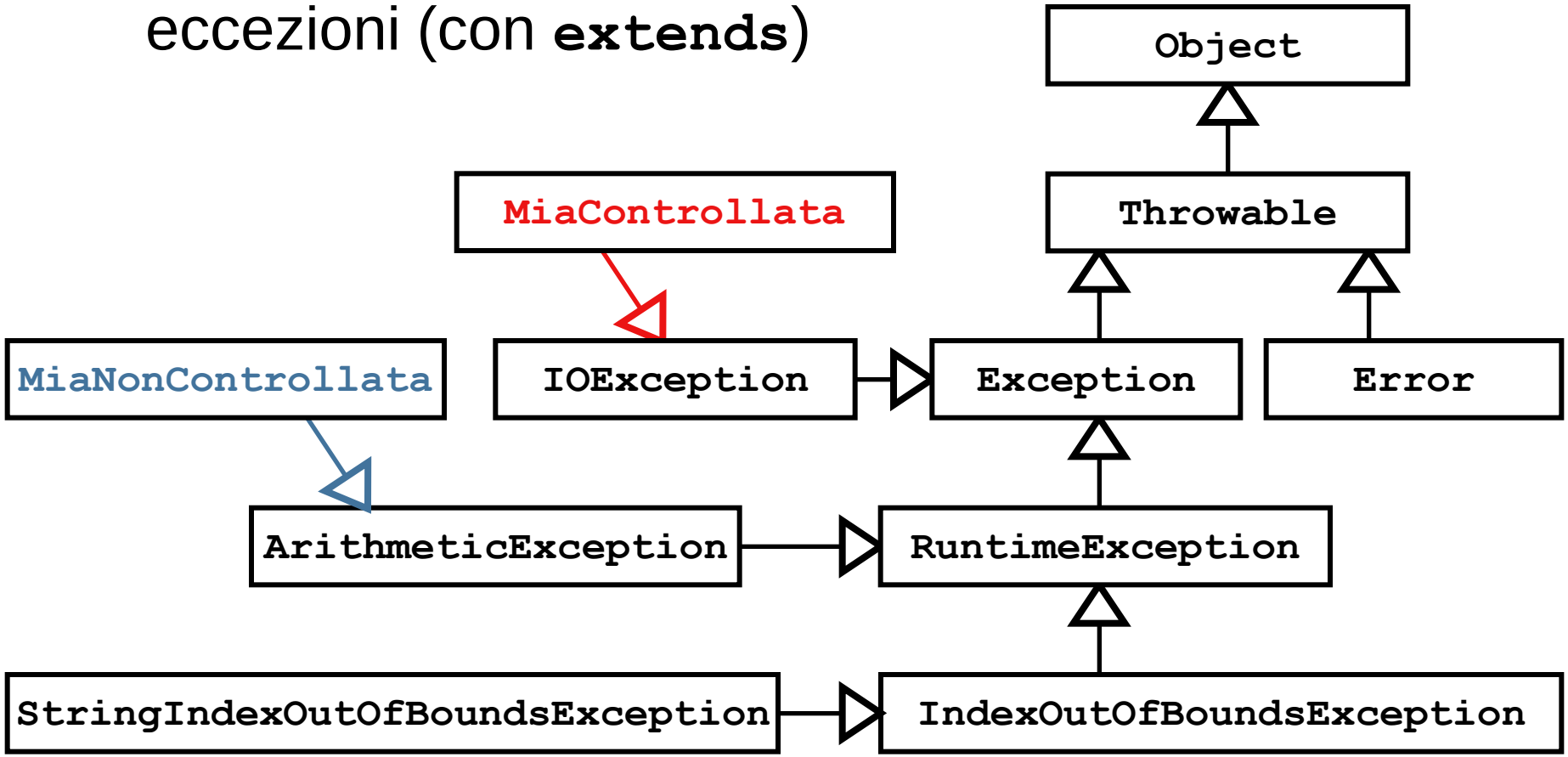
Eccezioni controllate

- Quelle derivanti soltanto da **`Exception`** sono dette **eccezioni controllate**.



Definire nuove eccezioni

- Si possono introdurre ulteriori eccezioni (con **`extends`**)



Definire nuove eccezioni

- Potrebbe essere utile scrivere nuove eccezioni che vanno sollevate se capita un certo evento indesiderato in un metodo scritto da noi.
  - Possiamo scriverle come estensioni di eccezioni già esistenti (quindi sfruttando l'ereditarietà)
- Scelta basilare tra eccezione controllata (*checked*) / non controllata (*unchecked*).
  - "Controllata" sta per "sottoposta a controlli da parte del compilatore" (discuteremo quali).
- Vediamo prima le eccezioni non controllate.

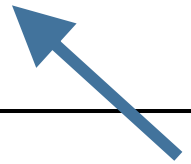
Definire eccezioni non controllate

- Per scrivere una nuova eccezione non controllata bisogna estendere anche la classe **`RuntimeException`** (non solo **`Exception`**).
  - Estendiamo un'eccezione predefinita che somigli.
  - Per un'eccezione da lanciare in **`calcolaQuoto`** potremmo estendere **`ArithmeticException`**, che a sua volta estende **`RuntimeException`** (e quindi anche **`Exception`**).
  - Per un **`getStanza(i)`** con **`i`** fuori **`MAPPA`** si può estendere **`IndexOutOfBoundsException`**

    `...`

Definire eccezioni non controllate

```
public class MyException extends ArithmeticException
{  //COSTRUTTORE
   public MyException(String msg)
   {   super(msg);   }
   ...
```



- il costruttore sfrutta l'ereditarietà: in questo modo l'inserimento del messaggio viene delegato al costruttore di **`ArithmeticException`**.
- **`MyException`** eredita poi metodi **`toString()`**, **`getMessage()`** e **`printStackTrace()`**: non c'è bisogno di scriverli da zero

Definire eccezioni non controllate

```
public class MyException extends ArithmeticException
{ //COSTRUTTORE
  public MyException(String msg)
  {  super(msg);  }
}
```

```
private static int calcolaQuoto(int a, int b)
{ if ( b==0 ) throw new MyException("Divisore 0!");
  else return a/b;
}
```

- **`calcolaQuoto`** può lanciare l'eccezione.

Limiti al lancio/cattura

- I blocchi di inizializzazione statica non possono né lanciare né catturare eccezioni.
- I metodi statici non possono lanciare eccezioni ma possono catturarle.
- I costruttori possono lanciare eccezioni, ma non possono catturarle.

- Infine, in Java è obbligatorio gestire le eccezioni controllate.

Eccezioni controllate

- Le eccezioni controllate sono tutte quelle eccezioni che estendono (direttamente o indirettamente) la classe **`Exception`** senza essere sottoclassi (dirette o indirette) della classe **`RuntimeException`**.
- Si chiamano così perché il compilatore controlla che vengano gestite esplicitamente:
  - o intercettate tramite **`try – catch`**
  - o rimandate al chiamante tramite **`throws`**

La clausola **throws**

- Le eccezioni controllate non possono essere propagate a piacere dai metodi.
- Un metodo che vuole poterla invocare deve annunciarlo esplicitamene nell'intestazione, usando la parola chiave **throws**.

```
tipo_rit metodo(param) throws tipo_Throwable
{ /* corpo del metodo ...*/ }
```

- La **throws** nell'intestazione deve essere seguita da un'istanza di **Throwable** (cioè di classe **Throwable** o di una sottoclasse).

Assenza di **`throws`**

- Se non viene annunciata, un'eccezione controllata non può essere lanciata.
- Quindi se manca la **`throws`**, le uniche sottoclassi di **`Throwable`** che si possono lanciare sono:
  - **`RuntimeException`** (e sue sottoclassi), ovvero, le eccezioni non controllate
  - **`Error`** (e sue sottoclassi) che però non sono eccezioni, bensì errori

Eccezioni controllate

- Supponiamo di avere un'eccezione controllata, correttamente annunciata da **`throws`**.
- A seguito di tale annuncio, il compilatore è in grado di fare questo controllo: se un metodo ne invoca un altro che annuncia di poter lanciare un'eccezione controllata, occorre che il metodo chiamante:
  - o catturi in ogni caso questa eccezione
  - o abbia dichiarato anch'esso di poterla lanciare

Eccezioni controllate

- In altre parole, il metodo chiamante:
  - o racchiude il metodo che annuncia la **`throws`** di un'eccezione controllata dentro **`try-catch`**, gestendo l'eventuale lancio dell'eccezione;
  - o propaga l'eccezione all'indietro, ma può farlo solo se pure lui l'ha annunciata in una **`throws`**.
- Il compilatore controllerà che ogni invocazione di un metodo che annuncia un'eccezione, o sia circondata da un **`try-catch`**, o avvenga in metodi che annunciano la stessa eccezione (o una sua superclasse).

Eccezioni controllate

- All'atto pratico, se stiamo scrivendo un metodo in cui vogliamo invocarne un altro, ma questo metodo annuncia una **`throws`** di un'eccezione controllata che non sappiamo gestire:
  - o ci sforziamo e scriviamo un **`try-catch`** in cui si gestisca l'eccezione;
  - o inseriamo questa eccezione nelle nostre **`throws`**, con l'idea di propagare l'eccezione
  - oppure, mettiamo lo stesso l'invocazione in un try-catch, e propaghiamo **un'altra eccezione** (una di quelle che abbiamo annunciato)

Definire eccezioni controllate

```
public class MyException extends ArithmeticException
{ //COSTRUTTORE
  public MyException(String msg)
  {  super(msg);  }
}
```

ora è un'eccezione controllata

qua compila correttamente

```
private static int calcolaQuoto(int a, int b)
{ if ( b==0 ) throw new MyException("Divisore 0!");
  else return a/b;
}
```

qua invece dà errore in compilazione

- Il compilatore lamenta l'assenza di una **`throws`** che annunci il lancio di **`MyException`**.

Definire eccezioni controllate

```
private static int calcolaQuoto(int a, int b)
        throws MyException
{ if ( b==0 ) throw new MyException("Divisore 0!");
  else return a/b; }
```

OK

- Adesso però il problema è il **`main()`**!

```
public static void main(String[] args)
{ ...
  System.out.print("Inserisci il divisore: ");
  int div2 = in.nextInt();
  int quoto = calcolaQuoto(div1,div2);
  System.out.println("Quoziente = " + quoto);
}
```

qua non c'è **`throws`**

e qua non c'è **`try..catch`**

- Il compilatore controlla di avere almeno una delle 2

Controllate vs. non controllate

- La clausola **`throws`** non deve per forza precedere eccezioni controllate.
  - Si può usare anche per eccezioni non controllate, sebbene non sia richiesto dal compilatore.
  - Ha comunque senso per documentare il codice.
- Le eccezioni annunciate nella **`throws`** sono automaticamente documentate da **`javadoc`**.
  - Ma la documentazione di default è vuota. Per inserire qualcosa bisogna usare **`@throws`**

Controllate vs. non controllate

- Come decidere se controllare le eccezioni?
- **Non controllate**: descrivono eventi che non dovrebbero verificarsi in programmi ben scritti.
  - Non serve controllarle, tanto tramite opportuni controlli si possono evitare
- **Controllate**: descrivono eventi che, per quante precauzioni si prenda, possono comunque capitare, così si cerca di tracciarli
  - Sono eventi esterni al programma: errori di lettura file, no connessione di rete... e **clonazione**!