DIETI UNINA UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea in Ingegneria Informatica

Corso di Ingegneria del Software

Gestione delle eccezioni in Java

Riferimenti

- Bruce Eckel, “Thinking in Java”: capitolo 9

::. Anomalie (1/2)

- Anomalia (o anche errore): Condizione anormale e inattesa che si manifesta durante l'esercizio di un sistema
- Un errore può condurre al fallimento del sistema, cioè una deviazione dal comportamento atteso
  - *value failures*, per es. risultati non corretti prodotti dal sistema
  - *timing failures*, per es. attesa indefinita, arresto del sistema
- Le anomalie devono essere opportunamente gestite dal programmatore

::. Anomalie (2/2)

- Alcune frequenti cause di anomalie:
  - Memory errors (i.e. memoria non allocata correttamente, memory leaks, “null pointer”)
  - File system errors (i.e. disk full, file non presente)
  - Network errors (i.e. network disconnessa, URL inesistente)
  - Computation errors (i.e. divisione per zero 0)
  - Array errors (i.e. accesso ad un elemento in posizione –1)

Un primo approccio – 1/4

Idealmente le condizioni anomale vanno previste dal programmatore.

Esempio (linguaggio C):

```
FILE* temp = fopen("test.txt", "r");
fscanf(temp, "%s", mystring);
```

Cosa succede se il file test.txt non esiste?

Il programma termina in maniera anomala.

Si richiede che il blocco di codice che ha generato il fallimento restituisca delle informazioni su quanto è accaduto a chi lo ha invocato, in modo che si possa agire per gestire opportunamente l'evento (eccezionale), evitando che il programma termini in maniera anomala.

Un primo approccio – 2/4

In linguaggi come il C, una condizione anomala è gestita da "codici d'errore" restituiti dalla funzione:

- un valore di tipo booleano (flag);
- uno tra *n* valori opportunamente codificati
  - es.: -1 = file_non_trovato,
    -2 = privilegi_insufficienti,
    etc.

Un primo approccio – 3/4

La procedura chiamante deve controllare il valore restituito dalla funzione per conoscerne l'esito dell'esecuzione.

```
bool apri_file(char* nome){
  FILE* temp = fopen(nome, "w");
  …
  if (!temp) return false;
  else return true;
}
```

```
int main(){
 …
 if (!apri_file("myFile.txt")) {
  print("Non posso aprire il file");
  exit;
 }
 …
}
```

Un primo approccio – 4/4

Tale approccio è poco efficiente:

- il controllo di eventi non standard è completamente a carico del programmatore attraverso la verifica dei valori di ritorno (cosa che non sempre viene fatta!);
- un semplice *flag* booleano contiene poche informazioni al fine di comprendere e poter gestire l'evento eccezionale;
- se i possibili valori di ritorno sono molti la procedura chiamante si “appesantisce”.

Eccezioni

- Una soluzione è di tipizzare le condizioni anomale, e forzare il programmatore a gestire le anomalie. Ciò è realizzato per mezzo di **`eccezioni`**.
- Una **`eccezione`** esprime una condizione anormale che si genera in una sequenza di codice durante l'esecuzione
- Tale situazione pregiudica la regolare esecuzione e richiede un'opportuna gestione

Meccanismo delle eccezioni

All'occorrenza di una anomalia (eccezione), occorre gestire una alterazione del flusso di controllo del programma.

✦Un **meccanismo delle eccezioni** è una struttura di controllo di un linguaggio, che consente di programmare le istruzioni cui deve essere ceduto il flusso di controllo del programma all'occorrenza della anomalia (gestione della eccezione), e decidere da dove riprendere il flusso di controllo al termine della sua gestione.

Gestione delle eccezioni – 1/2

- Il meccanismo di gestione delle eccezioni fornisce un'alternativa alle tecniche tradizionali che sono inefficienti e inleganti.
- Viene esplicitamente separata la parte di codice che si occupa della gestione delle eccezioni dal codice ordinario, così da rendere il programma più leggibile.

Gestione delle eccezioni – 2/2

- In un linguaggio ad oggetti, un'eccezione può verificarsi nel corso dell'esecuzione di un metodo.
  - A sua volta, tale metodo sarà stato invocato da un altro metodo
    - Tranne che per il metodo main, che è il punto di inizio del flusso di controllo del programma, ed è a sua volta invocato dal sistema operativo (ovvero dalla JVM per programmi Java)
- L'eccezione può essere gestita in due modalità:

- **Diretta**: il metodo che ha osservato/ricevuto l'eccezione la gestisce;
- **Indiretta**: il metodo che ha osservato/ricevuto l'eccezione non la gestisce, ma la "rinvia" al (metodo) chiamante.

Gestione delle eccezioni in Java

Tipologie di Java Exceptions

- **Unchecked Exceptions**
  Non è richiesto che questi tipi di eccezioni siano catturati o dichiarati in un metodo. Esempi sono:
  - *Runtime exceptions* che possono essere generate dai metodi o dalla JVM.
  - *Errors* che sono generati dall'interno della JVM e che spesso indicano un vero e proprio stato fatale dell'esecuzione

- **Checked Exceptions**
  Devono essere catturate da un metodo o dichiarate nella sua firma.
  - Aumentano la robustezza del codice a situazioni impreviste

Lanciare e dichiarare un'eccezione *checked*

- Java consente di lanciare intenzionalmente un'eccezione mediante l'istruzione throw
- Se un metodo solleva eccezioni, i loro tipi vanno elencati nella firma del metodo, preceduti dalla parola chiave throws

```
public void processFile(File f) throws IOException
{
    //...
    if (f.length() > MAX_LENGTH)
        throw new IOException("File too big");
    //...
}
```

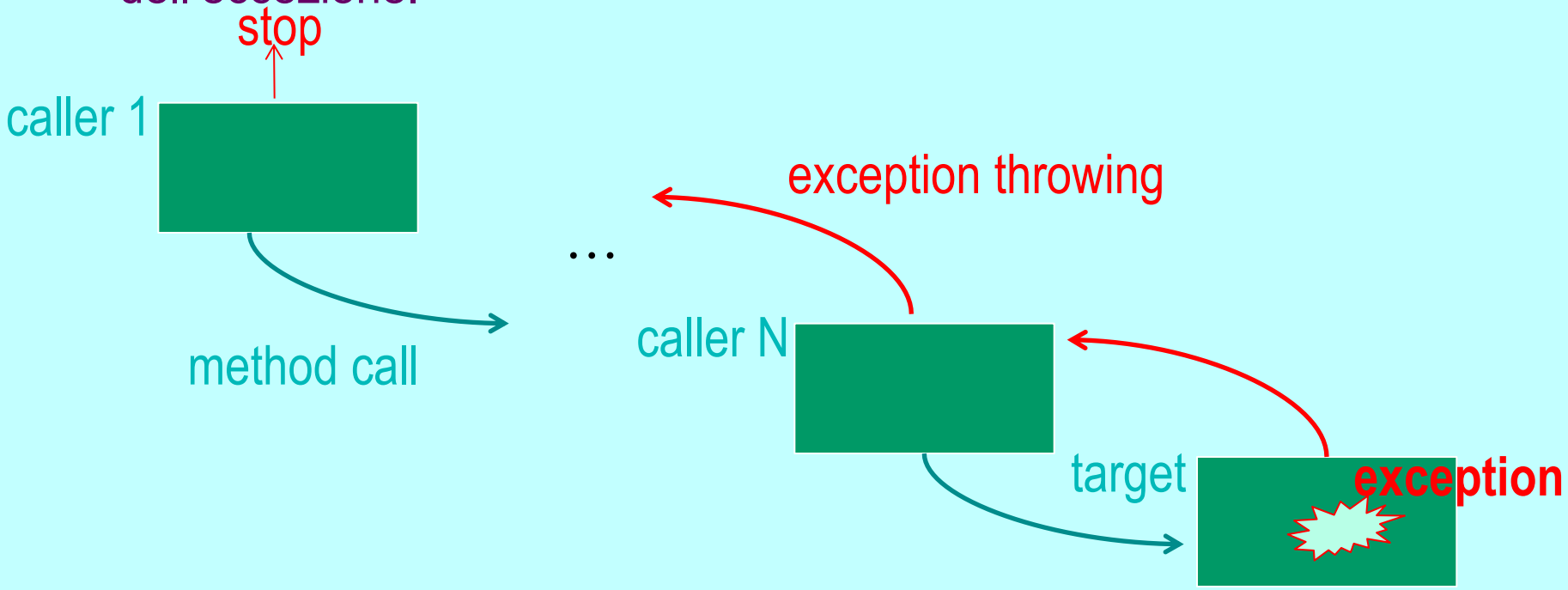
Lanciare e dichiarare un'eccezione *checked*

- Il compilatore può imporre (nel caso delle eccezioni *checked*) di specificare con throws che il metodo può sollevare un'eccezione

```
public void processFile(File f) throws IOException
{
    //...
    if (f.length() > MAX_LENGTH)
        throw new IOException("File too big");
    //...
}
```

Throwing di una eccezione

- Il processo di generazione o "rinvio" di una eccezione è detto **`throwing`** dell'eccezione.
  - Se non si gestisce l'eccezione il programma è interrotto e sono generate informazioni sullo stato del processo al momento dell'eccezione.

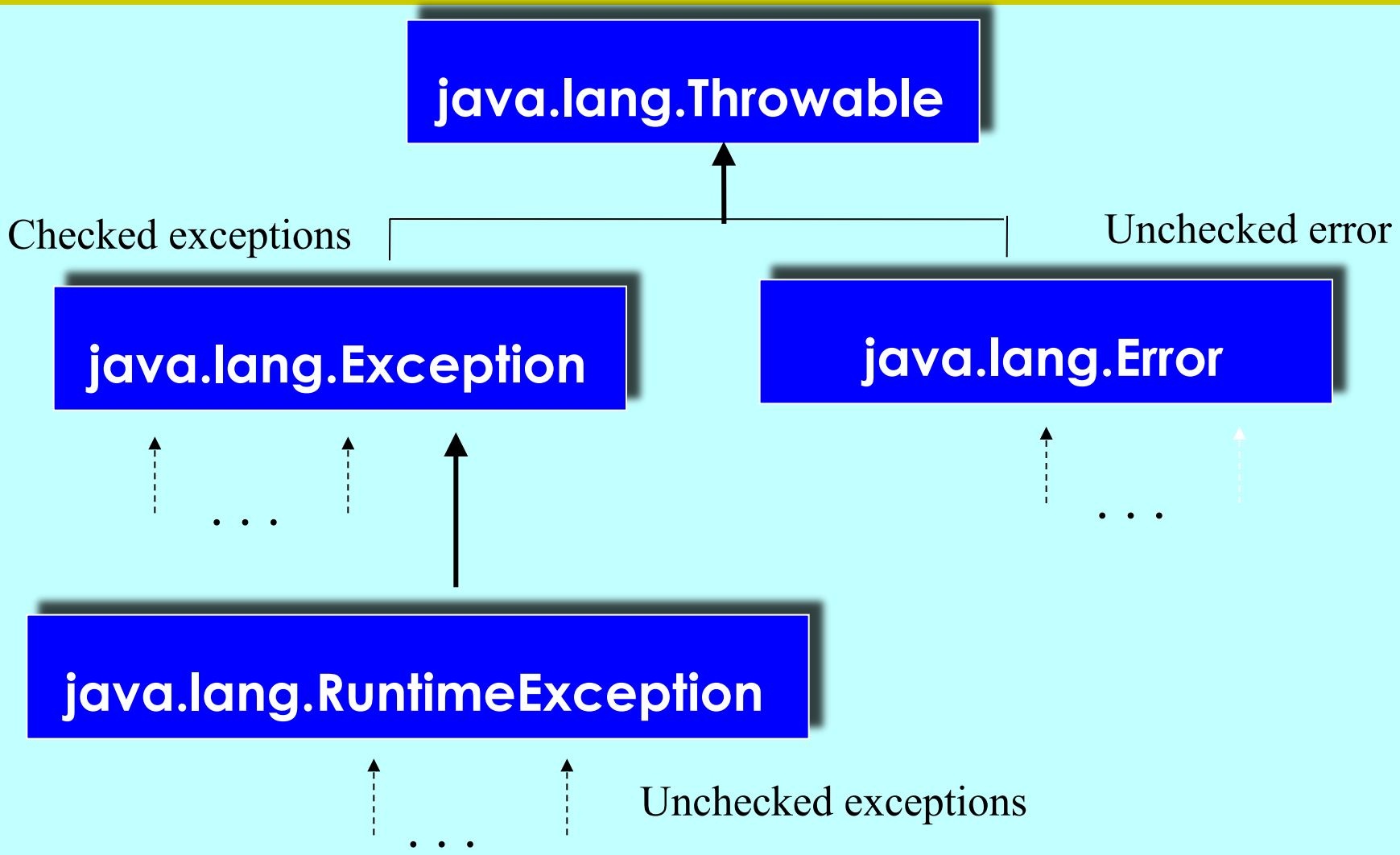


Throws

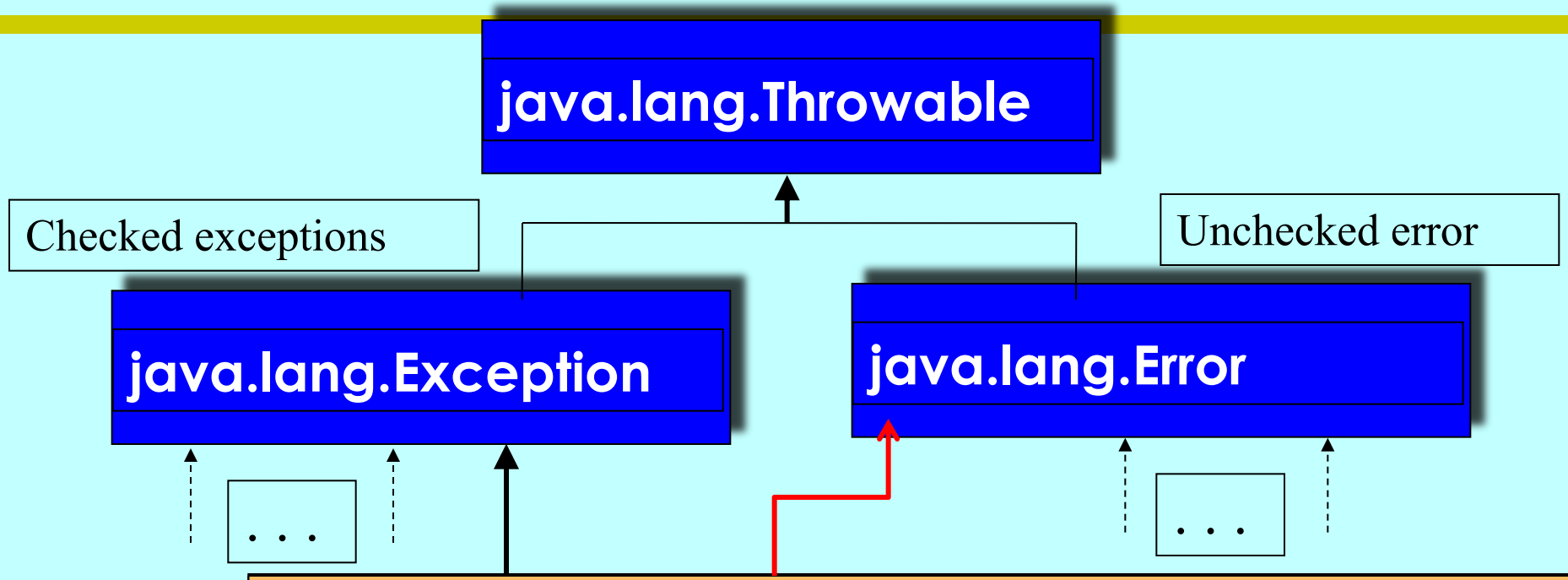
- In un blocco di codice che lancia un'eccezione si può decidere di gestire l'eccezione (*handling*) o di rilanciare l'eccezione al chiamante
- Per gestire l'eccezione si scrive un blocco try-catch.
- Per rinviare (la gestione del)l'eccezione al chiamante, si inserisce la clausola throws nella dichiarazione del metodo

```
public void myMethod throws IOException {
      //… codice con operazioni di I/O
}
```

Classificazione delle eccezioni



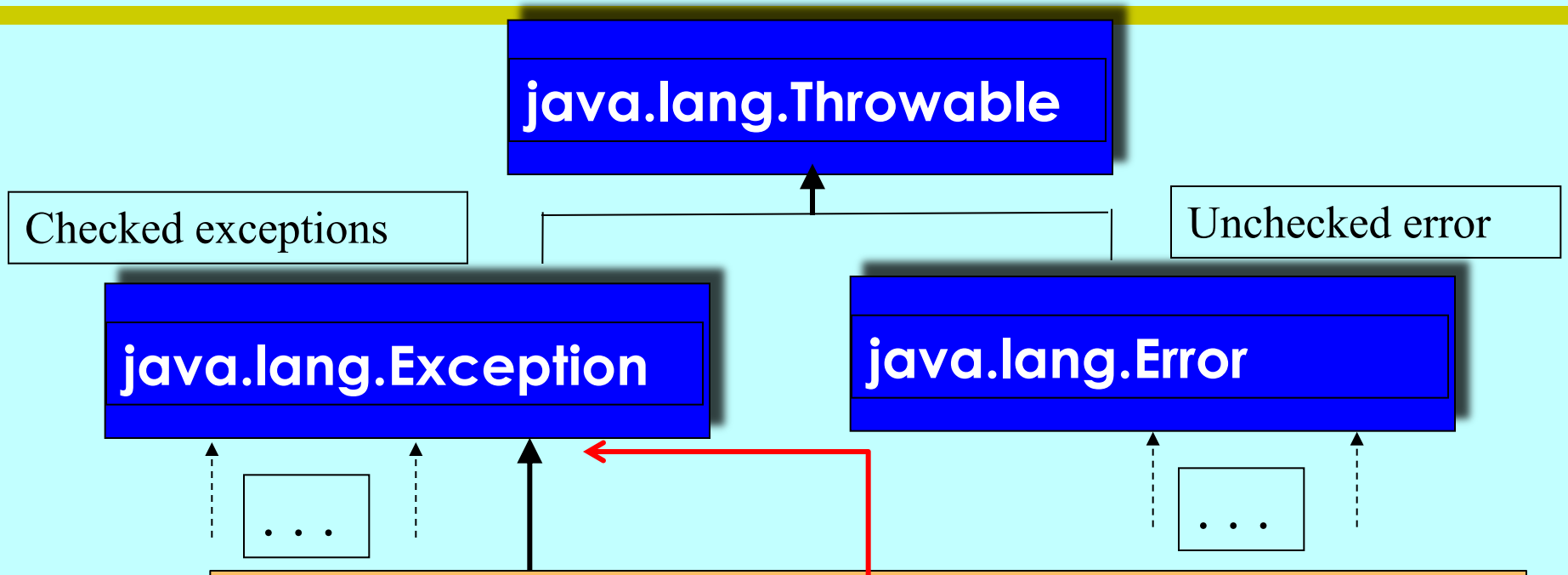
Classificazione delle eccezioni



`java.lang.Error:` descrivono problemi inusuali, di cui la procedura di ripristino è difficoltosa.

Generalmente sono dovuti all'insufficienza delle risorse oppure a gravi errori di programmazione.

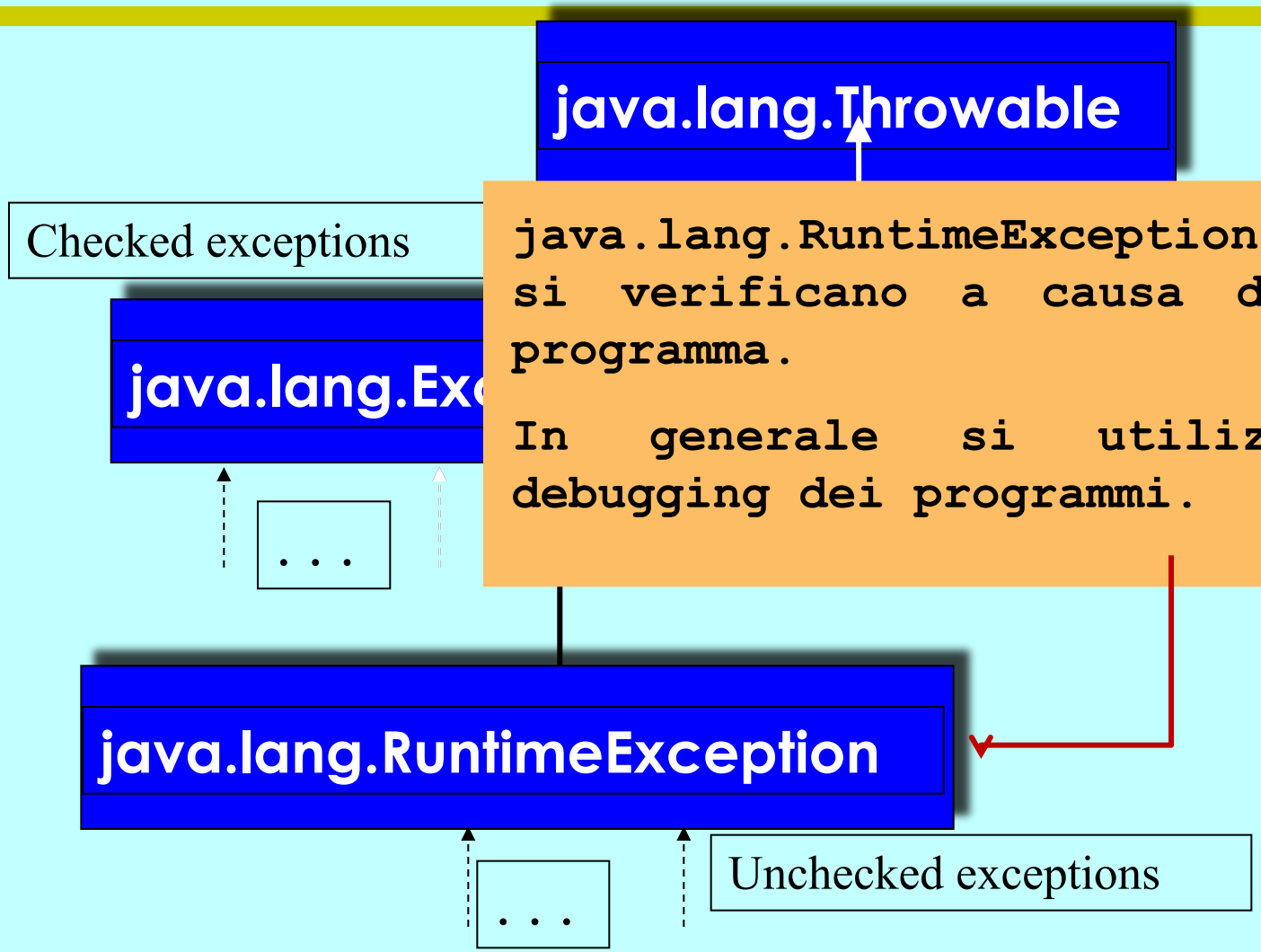
Classificazione delle eccezioni



`java.lang.Exception`: descrivono problemi che si possono verificare nell'esecuzione di un'istruzione, dovuti ad imprevisti:

- problemi di I/O
- errori dovuti ad altri componenti

Classificazione delle eccezioni



java.lang.RuntimeException: tipicamente si verificano a causa di "bug" del programma.

In generale si utilizzano per il debugging dei programmi.

Dettagli sul funzionamento

La classe Exception e la classe Error estendono la classe Throwable

All'occorrenza di una anomalia, la JVM instanzia un oggetto della classe eccezione relativa al problema (es. IOException), e lancia l'eccezione appena instanziata (con un throw).

Es.: divisione per zero. La JVM istanzierà un oggetto di tipo ArithmeticException (inizializzandolo opportunamente) e eseguirà il throw.
E' come se la JVM eseguisse le seguenti righe di codice:

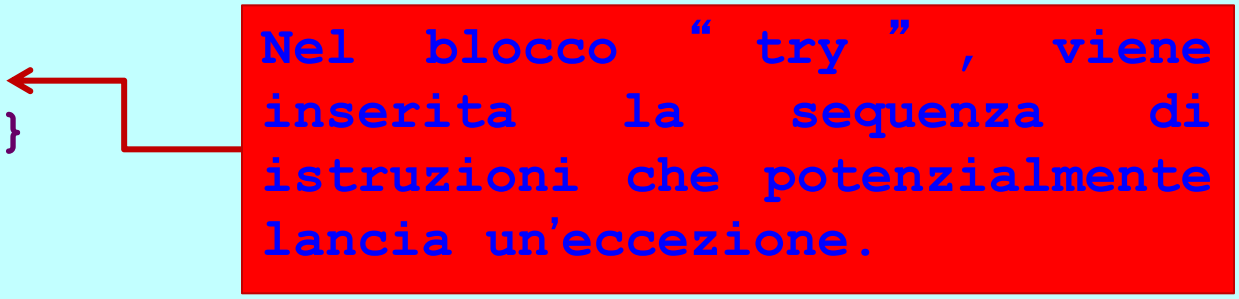
```
ArithmeticException exc = new ArithmeticException();
throw exc;
```

Esempi

- IndexOutOfBoundsException:
  - Relativa a array, string, e vector
- ArrayStoreException:
  - Quando si assegna un oggetto di un tipo non corretto ad un elemento di un array
- NegativeArraySizeException:
  - Quando si usa una dimensione negativa di un array
- NullPointerException:
  - Quando si fa riferimento ad un oggetto non istanziato
- SecurityException:
  - Quando la security viene violata, sollevata dal security manager

Try, catch, finally

- La gestione delle eccezioni si basa sul meccanismo try–catch e la clausola throws nella dichiarazione di un metodo.
  - `try { }`
  - `catch { }`




```
try {
   …
   Istruzione;
   …
}
```

Se una eccezione è sollevata durante un blocco try,
il resto del codice
nel blocco try non è eseguito

Try, catch, finally

- try { }
- catch { }

Nel blocco “ catch ”, si inserisce il codice per il trattamento dell'eccezione (exception handler).

```
try {
   …
   Istruzione;
   …
} catch(TipoEccezione1 e1){
//trattamento eccezione 1
} catch(TipoEccezione2 e2){
//trattamento eccezione 2
}
```

Try, catch, finally

Se un'eccezione non è intercettata dai blocchi "catch", essa sarà inoltrata al metodo chiamante.

```
try {
   …
   Istruzione;
   …
} catch(TipoEccezione1 e1){
//trattamento eccezione 1
} catch(TipoEccezione2 e2){
//trattamento eccezione 2
}
```

Try, catch, finally

Se un'eccezione non è intercettata dai blocchi "catch", essa sarà inoltrata al metodo chiamante.

```
try {
   …
   Istruzione;
   …
} catch(TipoEccezione1 e1){
//trattamento eccezione 1
} catch(TipoEccezione2 e2){
//trattamento eccezione 2
}
```

Un blocco try può essere seguito da una successione di blocchi catch: in quale ordine collocarli?

Try, catch, finally

- Il flusso di controllo di un blocco try/catch dipende dalle eccezioni sollevate nel blocco `try;`
- Il blocco `finally` permette al programmatore di scrivere istruzioni che verranno eseguite incondizionatamente, a prescindere dal verificarsi di eccezioni

Try, catch, finally

- Nel caso sia sollevata un'eccezione, il blocco "`finally`" verrà eseguito dopo il "`try-catch`".
- Consente di realizzare operazioni di clean-up.

```
try {
   istruzione1;
   ……
   istruzionen;
}catch ( TipoEccezione1 e1){
      //trattemento eccezione 1
} catch…

} finally {
      //codice che viene eseguito
      //in ogni caso
}
```

Esempio (1/3)

```
public class Ecc1 {
      public static void main(String args[]) {
      int a = 10;
      int b = 0;
      int c = a/b;
      System.out.println(c);
      }
}
```

•Questa classe può essere compilata senza problemi, ma genererà un'eccezione durante la sua esecuzione, dovuto alla divisione per zero.

Esempio (2/3)

- La JVM dopo aver interrotto il programma produrrà il seguente output:

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
Ecc1.main(Ecc1.java:6)
```

- evidenziando:
  - il tipo di eccezione (java.lang.ArithmeticException)
  - un messaggio descrittivo (/ by zero)
  - il metodo che ha sollevato l'eccezione (at Ecc1.main)
  - il file in cui è stata sollevata l'eccezione (Ecc1.java)
  - la riga in cui è stata sollevata l'eccezione (:6)

Esempio (3/3)

MA… il programma è terminato prematuramente!

Utilizzando le parole chiave try e catch sarà possibile gestire l'eccezione in maniera personalizzata:

```
public class Ecc2 {
    public static void main(String args[]) {
    int a = 10;
    int b = 0;
    try {
        int c = a/b;
        System.out.println(c);
        } catch (ArithmeticException exc) {
              System.out.println("Divisione per zero...");
              exc.printStackTrace();
              //stesso output di come se l'eccezione non fosse
              // catturata, ma senza interrompere il programma
              }
        }
}
```

Eccezioni definite dall'utente

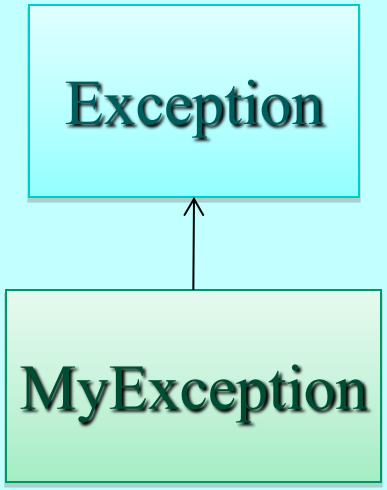
Il programmatore non è vincolato ad usare i tipi di eccezioni predefiniti, ma può crearne di propri.

Per creare un tipo di eccezione d'utente, il programmatore deve derivarla da un tipo esistente, preferibilmente quella dal significato più vicino alla nuova eccezione.

```
import java.lang.Exception;

class MyException extends Exception {
public MyException() {}
public MyException(String msg) {
super(msg);
}
}
```

Ordinamento dei blocchi catch



Supponiamo che il programmatore definisa un proprio tipo di eccezioni a partire dalla classe Exception di `java.lang`.

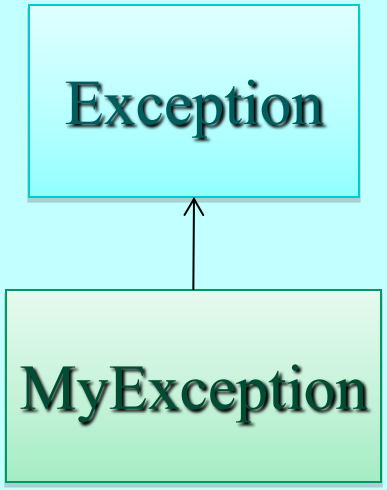
Supponiamo di avere il metodo

```
void do_smtg() throws MyException;
```

Consideriamo il seguente frammento di codice:

```
try{
…
do_smtg();
…
}catch(Exception ex) {…}
catch(MyException my_ex) {…}
```

Ordinamento dei blocchi catch



Supponiamo che il programmatore definisa un proprio tipo di eccezioni a partire dalla classe Exception di `java.lang`.

Suppo

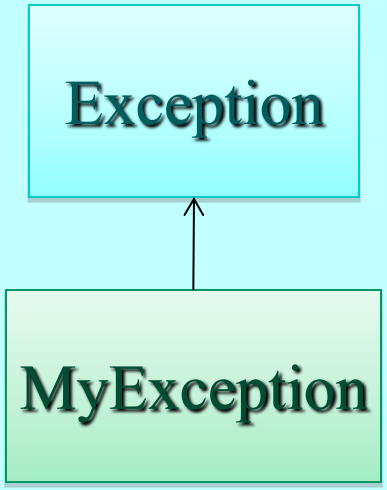
```
void do
```

Java tenta sempre di eseguire il primo blocco catch in grado di gestire l'eccezione appena sollevata. Il catch di Exception è utilizzabile per risolvere eccezioni di tipo MyException, quindi viene sempre eseguito, mentre l'altro mai.

Consideriamo il se

```
try{
…
do_smtg();
…
}catch(Exception ex) {…}
catch(MyException my_ex) {…}
```

Ordinamento dei blocchi catch



Supponiamo che il programmatore definisa un proprio tipo di eccezioni a partire dalla classe Exception di `java.lang`.

Supponiamo di avere il metodo

```
void do_smtg() throws MyException;
```

Consideriamo il seguente f

```
try{
…
do_smtg();
…
}catch(Exception ex) {…}
catch(MyException my_ex) {…}
```

Pertanto l'ordine dei blocchi catch deve essere l'inverso dell'ordine di derivazione: il catch di eccezioni derivate deve precedere quelli di eccezioni base.