Programmazione in Java
Parte II

Lezione 5

Dott. Marco Faella

- **Meccanismo moderno per gestire le situazioni di errore**

- **Ad esempio, cosa fare se un costruttore riceve argomenti non validi?**
- **Un costruttore non restituisce un valore, quindi non può restituire un valore speciale**
- **Soluzione: sollevare (o, lanciare) un'eccezione**

```
if (x<0) throw new RuntimeException(“x non deve essere negativo”);
```

- **Una volta sollevata, un'eccezione interrompe l'esecuzione del metodo attuale, e restituisce il controllo al metodo chiamante**
- **Se il metodo chiamante non *cattura* l'eccezione, essa interrompe la sua esecuzione e restituisce nuovamente il controllo al chiamante, fino ad arrivare al main**
- **Se neanche il main cattura l'eccezione, questa provoca la terminazione del programma**

Sollevare un'eccezione

- **Un metodo o costruttore può sollevare un'eccezione tramite l'istruzione throw**
- **La sintassi è la seguente:**

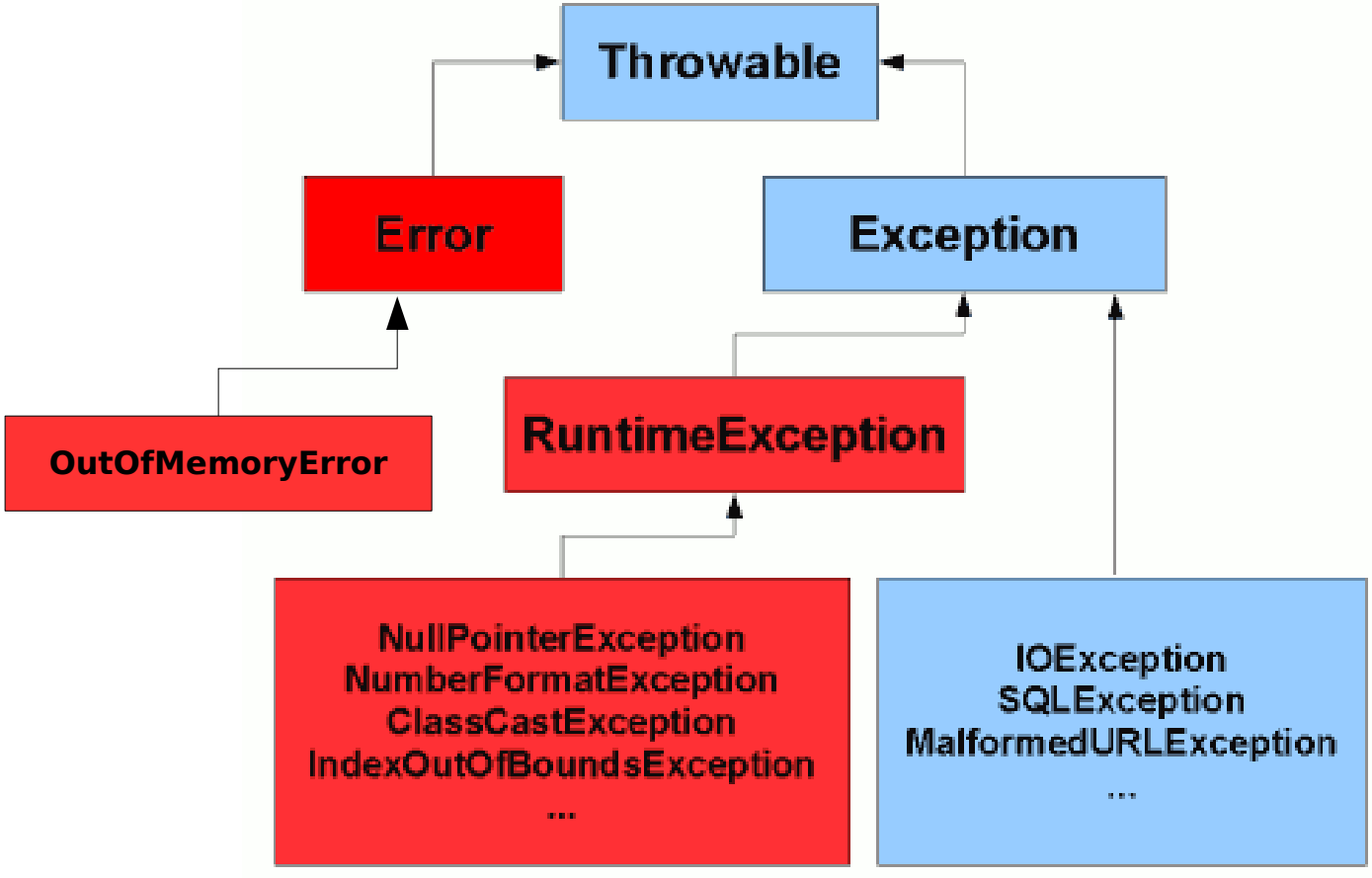
```
throw <oggetto di tipo eccezione>;
```

- **Tipicamente, l'oggetto eccezione viene creato sul posto tramite new**
- **Esercizio 1: rivedere quadrilatero**

- **La libreria standard Java offre un'ampia gamma di classi di eccezione, organizzate in una gerarchia**
- **La classe di più comune utilizzo è RuntimeException**
- **Tutte le classi standard di eccezione hanno due costruttori:**
  - **un costruttore senza argomenti**
  - **un costruttore che accetta una stringa, che rappresenta un messaggio d'errore**

- **Le eccezioni possono anche essere autonomamente sollevate dalla JVM**
- **Ad esempio:**
  - **quando si tenta di accedere ad un array con un indice non valido**
  - **quando si tenta di utilizzare un riferimento che punta a null**
- **Esercizio 2: accesso errato ad array**

La gerarchia delle eccezioni



Catturare le eccezioni

- **Un metodo può catturare un'eccezione con il costrutto try...catch**
- **La sintassi è la seguente:**

```
try {
   // codice che potrebbe sollevare un'eccezione
   ...
} catch (<classe di eccezione 1> e1) {
   // codice da eseguire se si verifica l'eccezione 1
   ...
} catch (<classe di eccezione 2> e2) {
   // codice da eseguire se si verifica l'eccezione 2
   ...
}
```

- **Se il codice all'interno del primo blocco solleva, direttamente o indirettamente, un'eccezione di tipo effettivo T, questa eccezione verrà catturata dal primo blocco catch la cui classe sia una superclasse di T**
- **Quindi, verrà eseguito il codice di quel blocco catch**
  - **la variabile che viene dichiarata nella clausola catch punterà all'oggetto eccezione che è stato lanciato**
- **Infine, l'esecuzione riprenderà dal codice che segue tutto il blocco try...catch**

Eccezioni verificate e non verificate

- **Le classi di eccezione si dividono in *verificate* (in Inglese, checked) e *non verificate* (unchecked)**
- **Il termine “verificata” si riferisce al fatto che il compilatore verifica che tali eccezioni siano opportunamente trattate dal programmatore (si veda dopo), mentre le eccezioni non verificate sono ad uso libero**

- **Intuitivamente, se in un metodo c'è un'istruzione che potrebbe lanciare un'eccezione verificata, o tale istruzione è contenuta in un blocco try...catch, oppure il metodo deve dichiarare che può lanciare quella eccezione**

- **Precisamente, se un metodo contiene un'istruzione “throw x”, oppure chiama un metodo la cui intestazione contiene la dichiarazione “throws x”, dove x è un'eccezione verificata, il compilatore controlla che sia rispettata una delle seguenti condizioni:**

  1) l'istruzione throw (oppure la chiamata) è contenuta in un blocco “try..catch” in cui una delle clausole catch è in grado di catturare x, oppure

  2) il metodo in questione contiene nell'intestazione la dichiarazione “throws y”, dove y è una superclasse di x

- **Naturalmente, delle quattro combinazioni che sono corrette per il compilatore, una non ha senso: che un metodo lanci direttamente un'eccezione con throw e subito la catturi con try...catch.**
- **Le eccezioni servono per segnalare una situazione anomala al chiamante, mentre in questo caso l'eccezione sarebbe usata solo per modificare il flusso di controllo all'interno di un metodo, cosa per la quale esistono apposite istruzioni (in particolare, if-then-else e break)**

Eccezioni verificate e non verificate

- **Con quale criterio si dovrebbe scegliere tra le due categorie di eccezioni?**

- **In linea di massima, dovrebbero essere di tipo verificato quelle eccezioni che il programmatore non può avere la certezza di evitare e che richiedono un opportuno trattamento**

- **Ad esempio, si consideri l'eccezione che viene lanciata quando si cerca di aprire un file inesistente (java.io.FileNotFoundException)**
- **Per quanto si sforzi, il programmatore non può raggiungere la certezza che tale eccezione non venga lanciata**
- **Difatti, anche se con una apposita istruzione si controllasse l'esistenza del file subito di prima di aprirlo, nei moderni sistemi multi-tasking niente può impedire ad un altro programma di cancellare il file in un momento intermedio tra il controllo di esistenza e la sua apertura vera e propria**
- **Pertanto, l'eccezione FileNotFoundException è verificata**

Eccezioni verificate e non verificate

- **Come ulteriore esempio, si consideri l'eccezione ArrayOutOfBoundsException, che viene lanciata dalla Java Virtual Machine (JVM) quando si tenta di accedere ad un array con un indice non valido**
- **Questa eccezione è evitabile da parte del programmatore e difatti indica la presenza di un errore di programmazione**
- **Inoltre, se tale eccezione fosse verificata, il programmatore dovrebbe dichiararla o trattarla in occasione di ogni accesso ad array**
- **Pertanto, ArrayOutOfBoundsException non è verificata**

Eccezioni verificate e non verificate

- **Infine, si consideri l'eccezione OutOfMemoryError, che viene lanciata dalla JVM se il progamma ha richiesto una allocazione di memoria (new), ma non vi è memoria libera a sufficienza**
- **Anche se tale situazione non è evitabile da parte del programmatore, c'è poco che il programmatore possa fare di conseguenza, tranne terminare il programma, che del resto è il comportamento di default in seguito al lancio di un'eccezione**
- **Inoltre, come per ArrayIndexOutOfBoundsException, le occasioni per il verificarsi di questa eccezione sono troppo frequenti per obbligare il programmatore a gestirla**
- **Pertanto, OutOfMemoryError è un'eccezione non verificata**

- **Per approfondire questo argomento, si veda anche l'articolo “Simple guide to checked exceptions”, di Gabriele Carcassi, facilmente rintracciabile sul web**

Eccezioni verificate e non verificate

- **Quali eccezioni della libreria standard sono verificate?**
- **Quale sarà la categoria di una nuova eccezione?**

- **Tutte le eccezioni della libreria standard sono verificate, tranne quelle che discendono da RuntimeException o da Error**
- **Quando ci crea una nuova classe di eccezione, tale classe eredita la categoria (verificata o non verificata) dalla sua superclasse**

- **In ciascuno dei seguenti contesti, dire se è opportuno usare un'eccezione verificata o non verificata**

- **In un metodo che calcola la radice quadrata, se l'argomento passato è negativo**
- **In un metodo che interagisce con una stampante, se la stampante ha esaurito la carta**
- **In un metodo che accetta un oggetto, se l'argomento passato è null**
- **In un metodo che chiude un file, se quel file non era stato precedentemente aperto**
- **In un metodo che accede ad un campo istanza privato, se quel campo si trova in uno stato non valido (cioè, contiene un valore non previsto)**