Il linguaggio Java

Le eccezioni

Eccezioni

- **Definizione**: evento *eccezionale* che si verifica durante l'esecuzione di un programma e che ne altera il normale flusso di esecuzione
- Situazioni che causano eccezioni
  - *Dati di input errati*
  - *Errori nei dispositivi: hard disk failure; network unavailability; stampante senza carta*
  - *Errori di programmazione: indice di un array non valido; accesso ad un riferimento di valore* ***null***
  - *Esaurimento delle risorse di sistema*
  - *altro*

Eccezioni

- Le eccezioni sono inevitabili
- Ciò che realmente importa è:
  ***cosa fare dopo che un errore si è verificato*?**

Esempio: Pila

- Pila di dimensioni limitate (stack_size)
- Invariante
  - #push ≤ #pop + stack_size
  - #pop ≤ push
- Una violazione dell'invariante causa un'eccezione

Soluzione senza le eccezioni

- MyStack

Esempio: Pila

Problema n. 1. *Caso normale e caso eccezionale non sono separati*

```
if (pila.push(elem) != -1)
     <caso normale>
else
     <caso eccezionale>
…
if ((elem = pila.pop()) != -1)
     <caso normale>
Else
     <caso eccezionale>
```


Esempio: Pila

Problema n. 2. Il valore ***-1*** *codifica "evento eccezionale"*

- Si attribuisce un significato speciale ad uno dei possibili valori ➔ non è una buona idea!
- Come gestisco **-1** (operazione **pop**) in una pila di **char** o di oggetti?

Esempio: la pila

Una soluzione migliore basata sulle eccezioni

Lancio e cattura

- ***Lancio di un'eccezione (throwing an exception)***
  - Creazione di un ***oggetto eccezione*** che viene passato al supporto run-time;
- ***Cattura di un'eccezione* (*catching the exception*)**
  - **Gestore** (**handler**) dell'eccezione
  - Il supporto run-time riceve un oggetto eccezione e trova il ***gestore*** (***handler***) a cui passarlo

Statement throw

- Un programma Java può utilizzare lo statement **throw** per lanciare un eccezione
- **throws *someThrowableObject*;**

Oggetto eccezione

- *L'**oggetto eccezione*** contiene informazioni riguardanti l'eccezione, tra le quali:
  - il ***tipo*** dell'eccezione
  - lo ***stato*** del programma quando l'eccezione si è verificata

Gestore delle eccezioni

- *Il blocco* **try**
  - racchiude le istruzioni che possono generare eccezioni
  - definisce l'ambito (*scope*) del gestore associato, cioè se un eccezione che viene lanciata nel blocco **try** viene gestita dal gestore associato

Gestore delle eccezioni

- *Il blocco* **catch**
  - permette di associare un gestore ad un blocco **try**
  - ad un blocco **try** possono essere associati uno o più gestori (blocchi **catch**)
  - ad un blocco **try** deve essere associato almeno un blocco **catch**

Ricerca del gestore

- Il supporto run-time cerca il ***gestore dell'eccezione***
  - ***A partire*** dal metodo in cui si è verificata l'eccezione,
  - ***A ritroso*** nei record di attivazione
  - Se nessun gestore viene trovato il programma termina
- Il supporto run-time cerca il gestore ***appropriato***
  - il tipo dell'eccezione è uguale al tipo, o a un sotto-tipo, dell'eccezione gestita dal gestore

Catch or throw

- *Un programma Java deve **catturare** o **rilanciare** le eccezioni controllate che lancia nel suo ambito*
  - *Eccezioni **controllate** lanciate nell'ambito di un metodo sono:*
    - *eccezioni che il metodo lancia direttamente per mezzo dello statement **throw***
    - *eccezioni lanciate indirettamente dai metodi chiamati dal metodo*
- *Un programma cattura un'eccezione fornendo un gestore dell'eccezione (***try-catch-finally***)*
- *Se un metodo non cattura un eccezione allora la deve rilanciare* (**throws**)
  - Di questo comportamento deve essere avvisato l'utente del metodo. Quindi tale eccezione viene specificata nell'interfaccia del metodo per mezzo della clausola **throws**

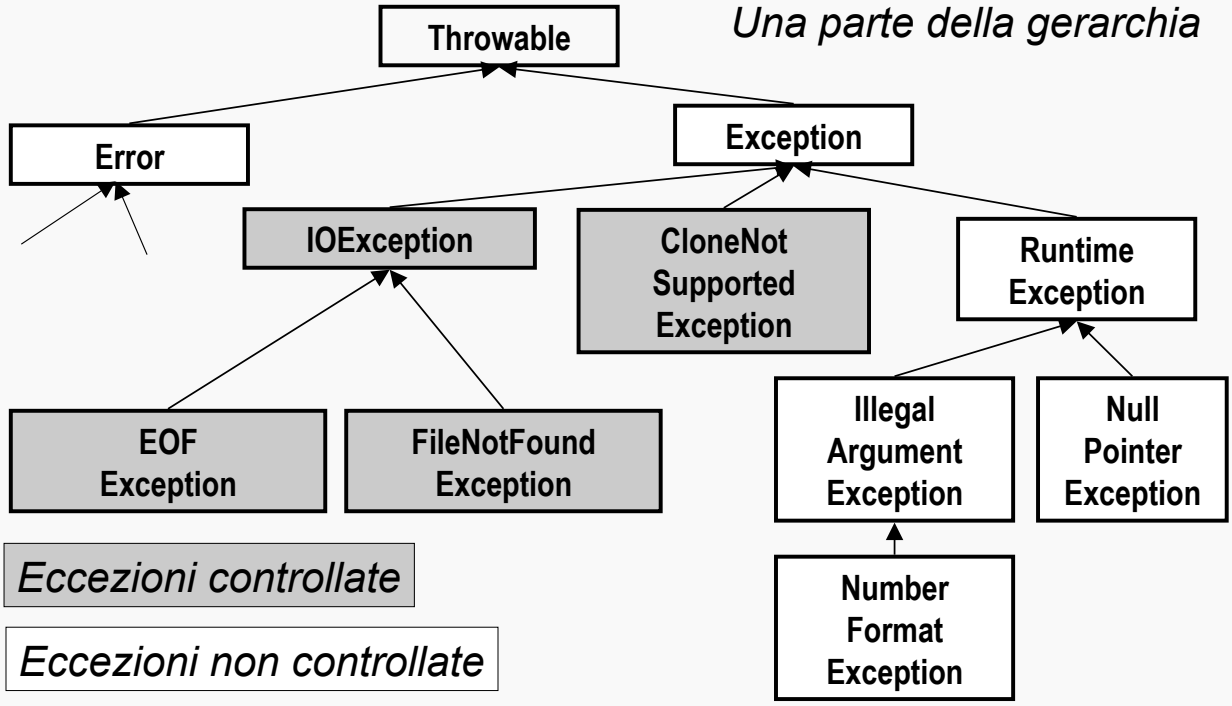
Gestione delle eccezioni

- Catturare eccezioni
  - Blocco **try-catch**
- Rilanciare (propagare) eccezione al chiamante
  - **Esempio.** La open di un file fallisce perché “file not found”
  - Clausola **throws**

Vantaggi delle eccezioni

- Separazione del codice per la gestione dell’errore dal codice “normale”
- Propagazione dell’errore fino al punto in cui interessa gestirlo
- raggruppamento e differenziazione degli errori

Gerarchia delle eccezioni



Le eccezioni derivate da **Error**

- Quando si verifica un errore “fatale” all’interno del supporto runtime, viene lanciata un’eccezione di tipo **Error**
  - *Ad esempio il collegamento dinamico fallisce*
- Tipicamente un programma Java non cattura né lancia un eccezioni di tipo **Error**.

Le eccezioni derivate da **Exception**

- Le **Exception** modellano errori non fatali
- Tipicamente, un programma Java cattura e/o lancia queste eccezioni
- Le **RuntimeException** sono lanciate da JVM
  - Esempi: **NullPointerException**, **ArithmeticException**, **IndexOutOfBoundsException**
  - *Queste eccezioni possono essere lanciate praticamente in ciascun punto di un programma. Perciò sarebbe troppo gravoso per il compilatore controllare che esse siano gestite. Perciò il compilatore permette che queste eccezioni possano essere né catturate e né rilanciate*
- Le ***eccezioni controllate*** sono eccezioni derivate da **Exception** che il compilatore controlla che siano rilanciate oppure catturate

Convenzioni sui nomi delle eccezioni

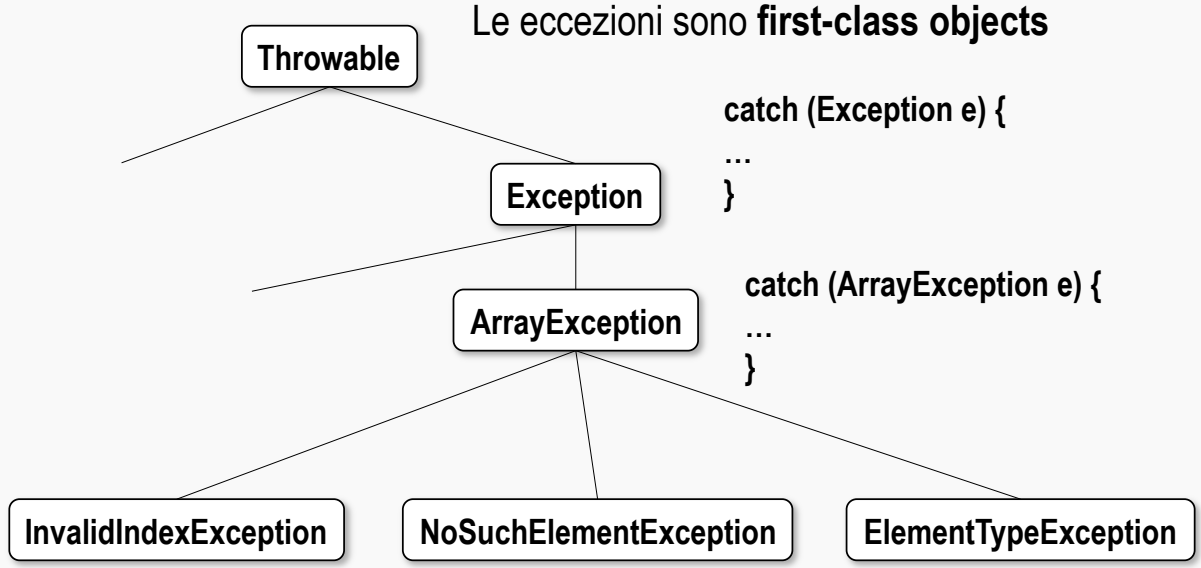
- **Come si sceglie il nome della classe eccezione?**
- La convenzione prevede che
  - Il nome termina in **Exception** se la classe deriva da **Exception**
  - Il nome termina in **Error** se la classe deriva da **Error**

Definire tipi di eccezioni

- **Come si sceglie la superclasse di MyStackException?**
- **Abbiamo tre possibilità**
  - **Throwable**: **sconsigliato** perché troppo generale
  - **Error**: **sconsigliato** poiché le eccezioni che derivano da **Error** si riferiscono ad errori fatali che si generano internamente al supporto runtime
  - **Exception**: è la scelta tipica.
    - Tipicamente è una buona scelta perché le sottoclassi di **Exception** sono o troppo specifiche o scorrelate da **MyStackException**
    - È sconsigliato che la classe **MyStackException** derivi da **RuntimeException** perché non è un vero errore del supporto runtime

Raggruppamento e differenziazione

Le eccezioni sono **first-class objects**




```
catch (Exception e) {
…
}
```

```
catch (ArrayException e) {
…
}
```

```
catch (InvalidIndexException e) {
…
}
```

Esempio: La pila

Handler per **push** ed handler per **pop**

MyStack con handler specifici

Blocco finally

- *Un gestore permette di catturare e gestire eccezioni*
- *Un gestore è costituito da tre componenti*
  - il blocco **try**
  - il blocco **catch**
  - il blocco **finally**

Il blocco finally

- *Il blocco* **finally**
  - fornisce un meccanismo che consente al metodo di eseguire operazioni di *cleanup* indipendentemente da cosa accade nel blocco **try**
  - Esempio: chiusura di un file; rilascio di risorse del sistema

Esempio: la pila

In ogni caso voglio fare la **pop** di tutti i dati

MyStack (con blocco finally)