
# Il linguaggio Java

## *Le eccezioni*

# *Shit happens!*

- Gli errori nei programmi si verificano
- Quello che è realmente importante è:
  *cosa fare dopo che un errore si è verificato*?

# Eccezione

- *Eccezione*

  *Evento eccezionale* che si verifica durante l'esecuzione di un programma e che 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

# Trattamento delle eccezioni

Quando si verifica un eccezione durante l'esecuzione di un metodo,

- ***lancio di un'eccezione (throwing an exception)***

  il metodo crea un ***oggetto eccezione*** e lo passa al supporto run-time;

- ***cattura di un'eccezione*** (***catching the exception***)

  il supporto run-time trova il codice ***gestore*** (***handler***) dell'eccezione e gli passa l'oggetto eccezione

# Trattamento delle eccezioni (II)

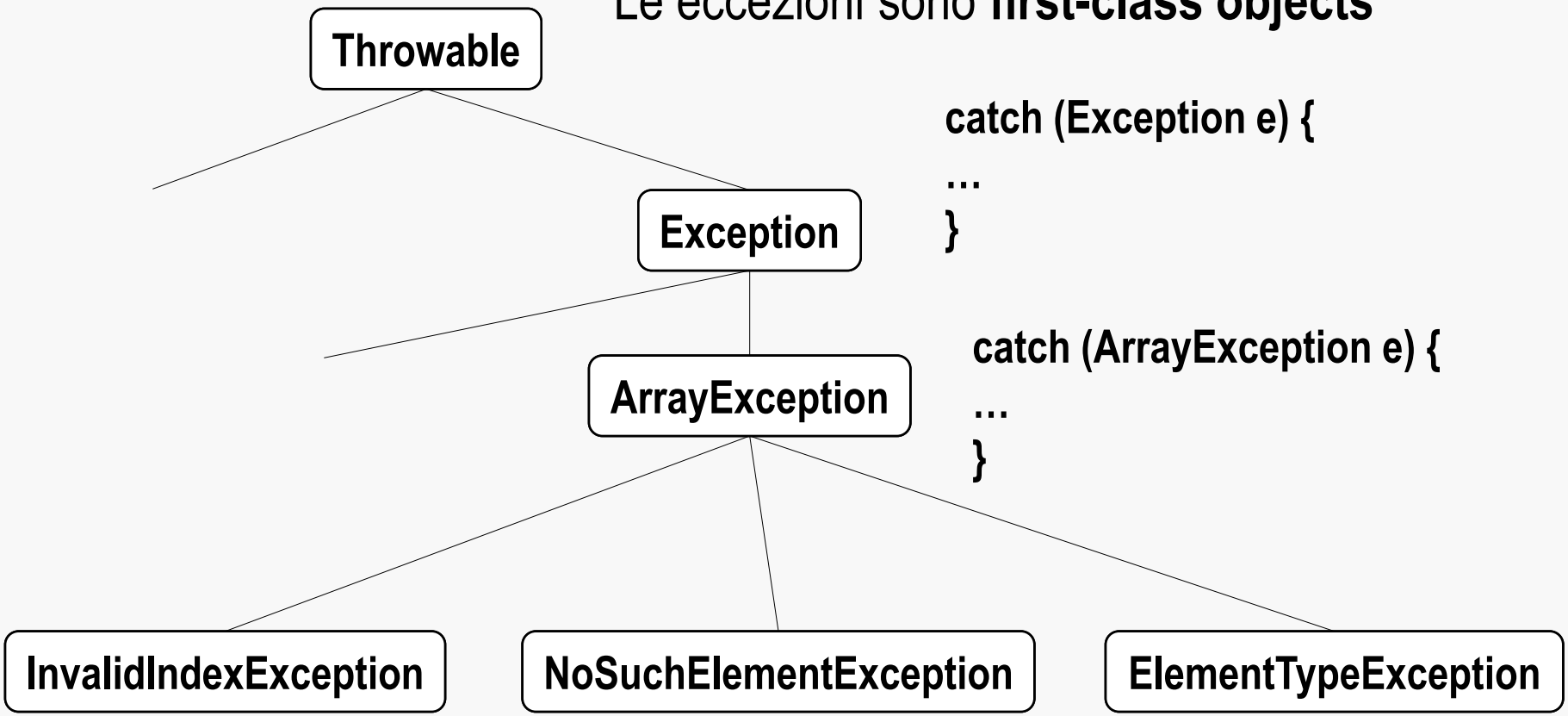
- *L'**oggetto eccezione*** contiene informazioni riguardanti l'eccezione, tra le quali:
  - il ***tipo*** dell'eccezione
  - lo ***stato*** del programma quando l'eccezione si è verificata
- Il supporto run-time cerca il ***gestore dell'eccezione***
  - ***a partire*** dal metodo in cui si è verificata l'eccezione,
  - ***a ritroso*** nei record di attivazione (se nessun gestore viene trovato il programma termina)
- Un gestore è considerato ***appropriato*** per un'eccezione se il tipo dell'eccezione è uguale al tipo, o a un sotto-tipo, dell'eccezioni gestite dal gestore

# Vantaggi

- *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*

# Raggruppamento e differenziazione

Le eccezioni sono **first-class objects**

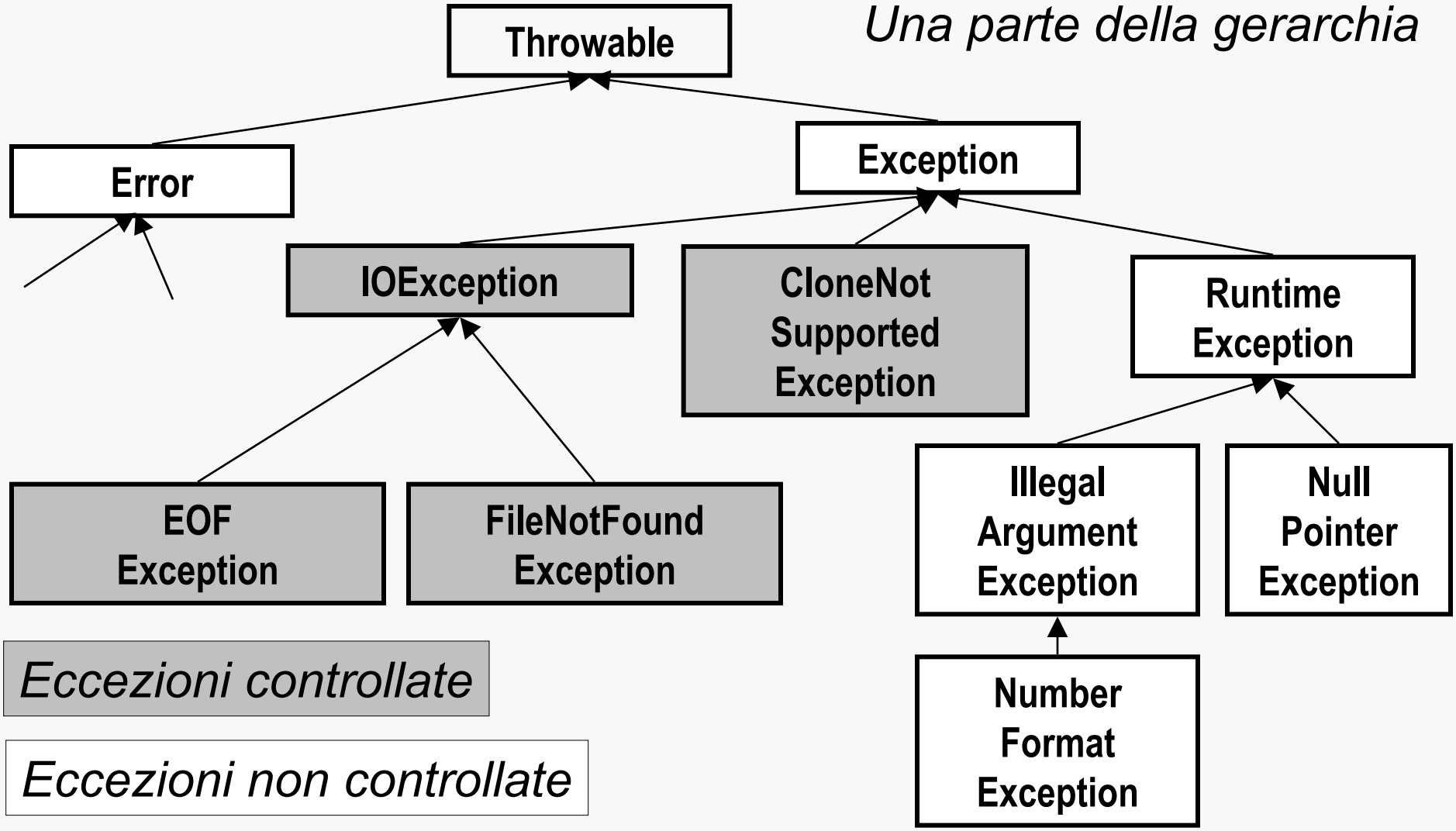



```
catch (Exception e) {
…
}
```

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

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

# Gerarchia delle eccezioni



# Le eccezioni derivate da Error

- Quando si verifica un errore “fatale” all’interno del supporto runtime, ad esempio il collegamento dinamico fallisce, viene lanciata un’eccezione di tipo **Error**
- 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

# Rilanciare le eccezioni


- In certi casi può essere adeguato catturare e gestire le eccezioni
- In altri casi può essere preferibile propagare l'eccezione al chiamante
  - Esempio. Si sta sviluppando un package e non siamo in grado di prevedere le esigenze degli utenti del package. In questo caso è preferibile che siano loro a gestire le eccezioni

# Statement throw

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

```
public Object pop() throws EmptyStackException {
   Object obj;

   if (size == 0) throw new EmptyStackException();

   obj = objectAt(size - 1);
   setObjectAt(size - 1, null);
   size--;
   return obj;
}
```

# Requisito catch & specify

- *Un programma Java deve **catturare** o **rilanciare** le eccezioni controllate che lancia nel suo ambito*
- *Un programma cattura un'eccezione fornendo un gestore dell'eccezione (**try-catch-finally**)*
- *Se un metodo non cattura un eccezione allora la deve rilanciare*
  - Di questo comportamento deve essere avvisato l'utente del metodo. Quindi tale eccezione viene specificata nell'interfaccia del metodo per mezzo della clausola **throws**
- 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

# Gestore delle eccezioni

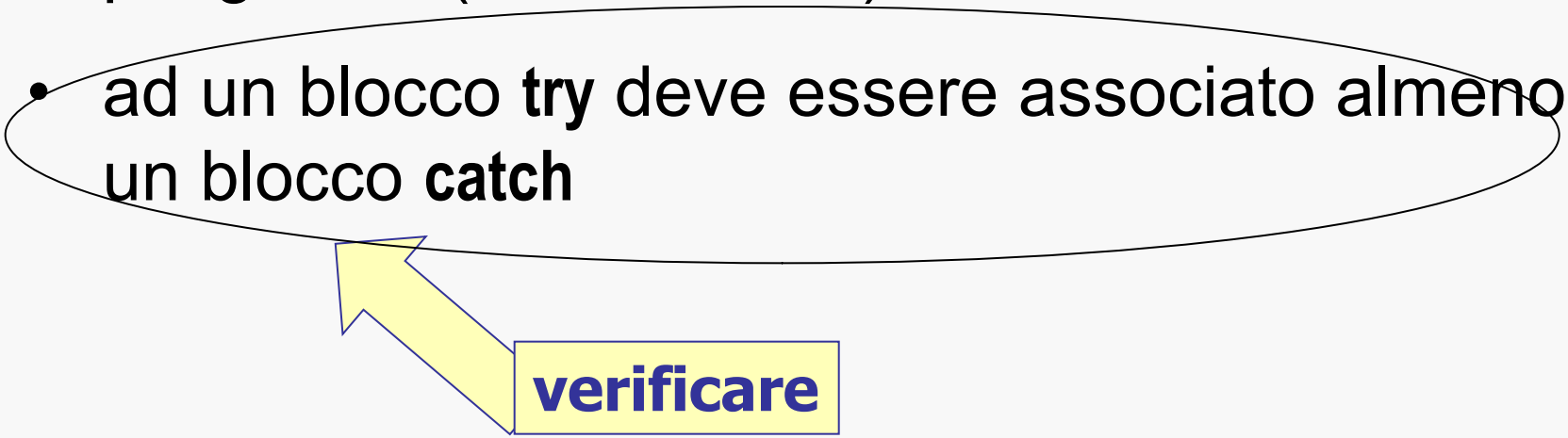
- *Un gestore permette di catturare e gestire eccezioni*
- *Un gestore è costituito da tre componenti*
  - il blocco **try**
  - il blocco **catch**
  - il blocco **finally**

# *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**



# Gestore delle eccezioni

- *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

# Il blocco try

```
try {
        statements
}
```

```
// blocco try per ListOfNumbers
PrintWriter out = null;

try {
    System.out.println("Entering try statement");
    out = new PrintWriter(new FileWriter("OutFile.txt"));
    for (int i = 0; i < size; i++)
        out.println("Value at: " + i + " = " + victor.elementAt(i));
}
```

# Il blocco catch

```
try {
        statements
} catch(…) {
        …
} catch(…) {
        …
}
```

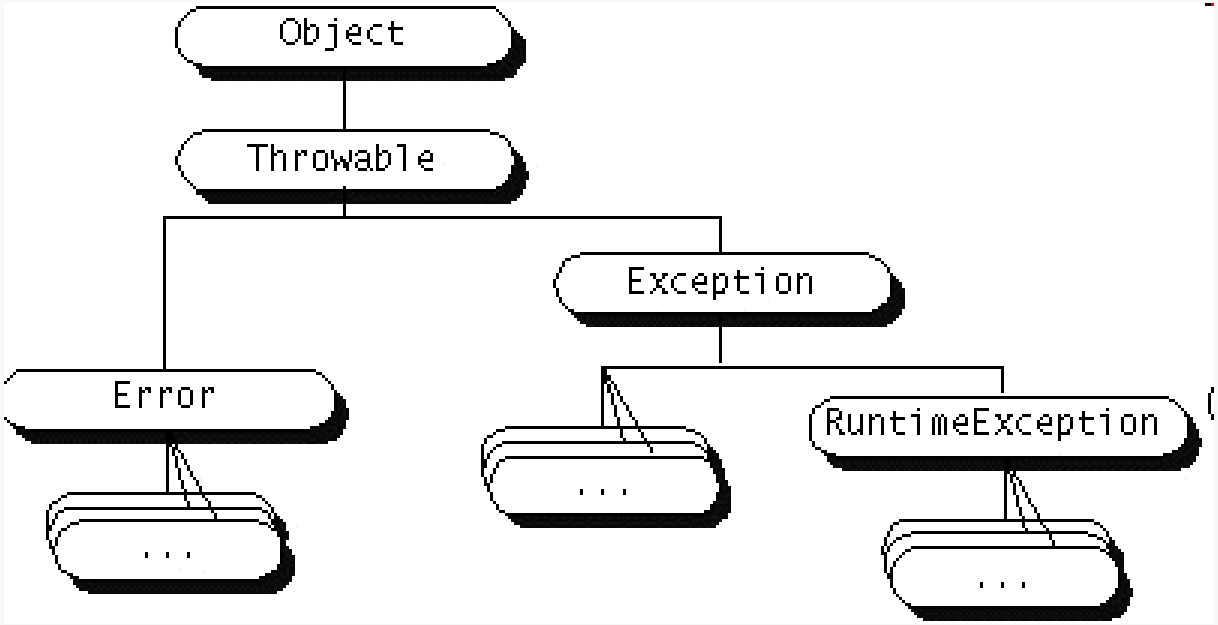
```
catch(SomeThrowableClass variableName) {
        statements
}
```

- ***SomeThrowableClass*** definisce la classe di eccezioni che il gestore può gestire
- Tale classe deve discendere da **Throwable**
- ***variableName*** è il nome con cui il gestore riferisce l'oggetto eccezione catturato
- Gli statement vengono eseguiti quando si invoca *l'exception handler*

# Il blocco catch

```
try {
    . . .
} catch (ArrayIndexOutOfBoundsException e) {
    System.err.println("Caught ArrayIndexOutOfBoundsException: " +
        e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " +
        e.getMessage());
}
```

# Gestori generali o specializzati?




- Un gestore generale gestisce un "nodo"
- Un gestore specializzato gestisce una "foglia"
- È bene che i gestori siano specializzati

# Il blocco finally

- Il blocco **finally** permette di eseguire operazioni di *cleanup* dello stato del metodo prima di passare il controllo (possibilmente) ad altre parti del programma
- Il blocco **finally** viene sempre eseguito indipendentemente da cosa accade nel blocco **try**

# *Il blocco finally per writeList*

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

# Ricerca ed esecuzione dell'handler


- Quando viene lanciata un eccezione, l'esecuzione del blocco **try** viene interrotta ed il supporto runtime tenta di trovare un gestore dell'eccezione come segue:
  - A partire dalla testa della *call stack* cerca un metodo che abbia un gestore per un'eccezione *legalmente assegnabile* a quella lanciata
  - Per ogni metodo, il supporto runtime controlla i gestori nell'ordine in cui appaiono dopo il blocco **try**
  - Se il supporto runtime non trova alcun gestore l'esecuzione del programma viene abortita

# Creare le proprie eccezioni

Quando di progetta un package bisogna

- progettare correttamente l'interazione tra le classi
- progettare correttamente le interfacce
- progettare correttamente le eccezioni che le classi possono lanciare
  - un utente può utilizzare in modo errato le classi
  - una chiamata legittima può produrre un risultato indefinito
  - una classe deve essere robusta, cioè
    - fare qualcosa di ragionevole con gli errori oppure
    - comunicare un errore al chiamante

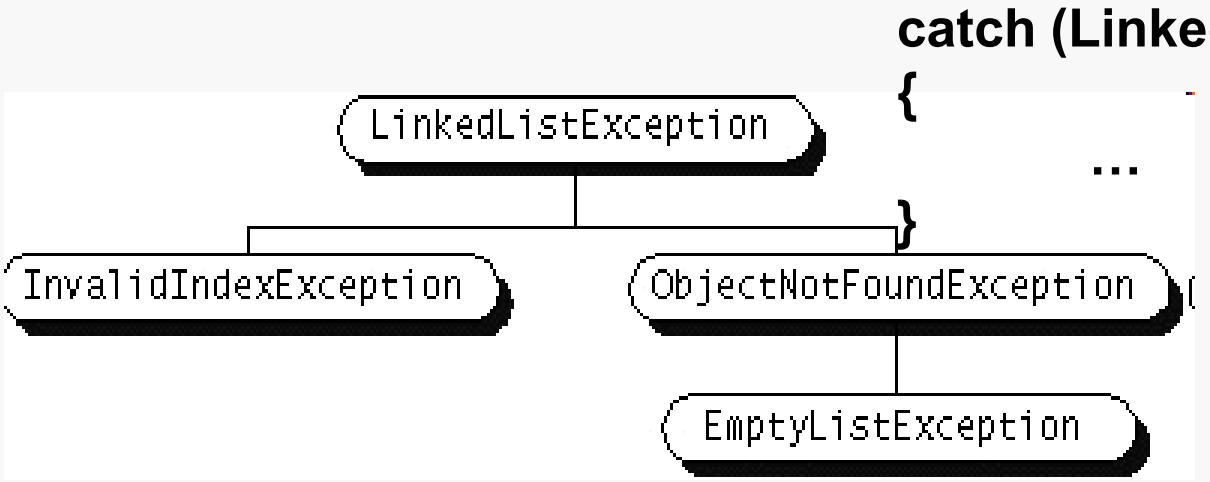
# Creare le proprie eccezioni (1)

- Supponiamo di progettare la classe **LinkedList** che supporta i seguenti metodi
  - **Object objectAt(int n)** che ritorna l'oggetto che occupa la posizione **n**-esima nella lista
  - **Object firstObject()** che ritorna il primo oggetto nella lista
  - **indexOf(Object o)** che cerca l'oggetto o nella lista e, se lo trova, ritorna la sua posizione
- Ciascun metodo può lanciare un eccezione in certe particolari condizioni e ciascun metodo può lanciare un eccezione diversa dagli altri
  - **Object objectAt(int n)** lancia un eccezione se **n** è minore di zero o maggiore o uguale del numero di elementi nella lista
  - **Object firstObject()** lancia un eccezione se la lista è vuota
  - **indexOf(Object o)** lancia un eccezione se l'oggetto specificato come parametro non è nella lista

# Creare le proprie eccezioni (2)

- **Quale classe di eccezione ciascun metodo deve lanciare?**
- **Ci sono due possibilità**
  - Riusare le eccezioni definite da qualcun altro, ad esempio quelle definite dall'ambiente di sviluppo di Java
  - Definire una propria classe di eccezioni
- **Criteri per scegliere l'una o l'altra possibilità**
  - È necessaria una classe eccezione non presente nell'ambiente di sviluppo di Java?
  - Potrebbe essere utilie agli utenti distinguere le proprie eccezioni da quelle di altri sviluppatori?
  - Il nostro codice lancia due o più classi di eccezioni correlate tra di loro?
  - Se si usano eccezioni di altri, queste sono accessibili all'utente?
  - Il proprio package deve essere auto-contenuto ed indipendente da altri package?

# Creare le proprie eccezioni (3)




- La classe **LinkedList** può lanciare più classi di eccezioni correlate tra di loro
- Potrebbe essere conveniente catturarle con un solo gestore, ma bisogna comunque permettere agli utenti di definire gestori specializzati
- Il package deve essere auto-contenuto

# Creare le proprie eccezioni (4)

- **Come si sceglie la superclasse di LinkedListException?**
- **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 **LinkedListException**
    - È sconsigliato che la classe **LinkedListException** derivi da **RuntimeException** perché non è un vero errore del supporto runtime

# Creare le proprie eccezioni (5)

- **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**