



Agent and Object Technology Lab
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma



Ingegneria del software A

Eccezioni

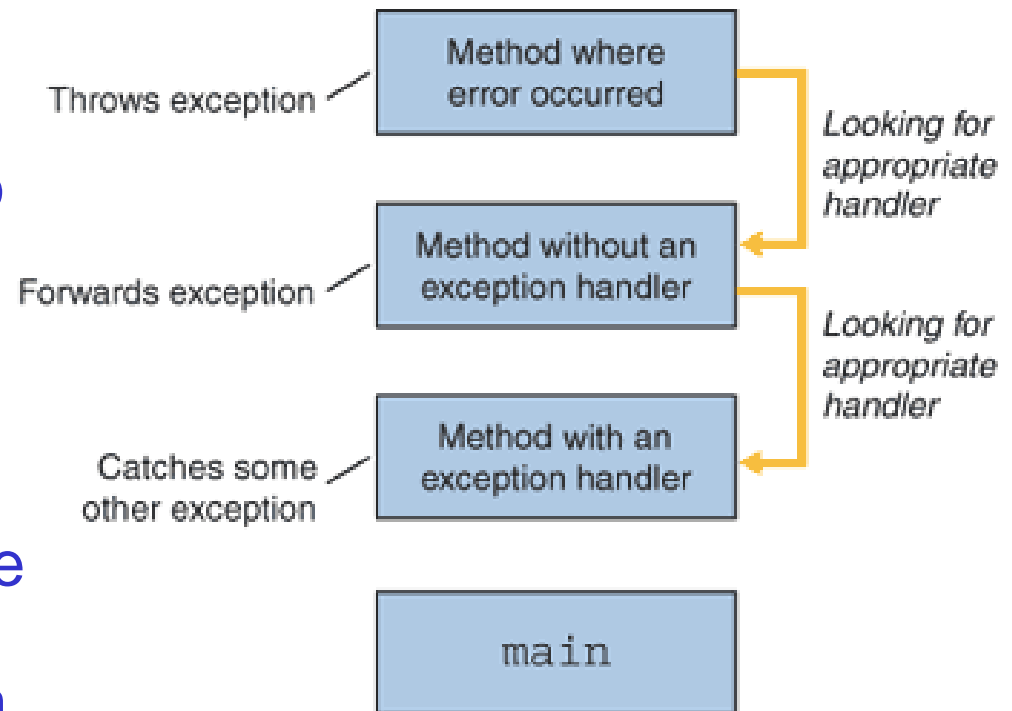
Prof. Agostino Poggi

- ♦ Il termine eccezione è un'abbreviazione per “*evento eccezionale*” e indica un evento verificato durante l'esecuzione di un programma che interrompe il suo normale flusso di istruzioni
- ♦ Quando si verifica un errore in un metodo, il metodo *lancia* (*throw*) un'eccezione
 - Il metodo crea un oggetto eccezione con informazioni sull'errore, il suo tipo e le condizioni in cui si è verificato
 - Lo consegna all'ambiente di *runtime* cerca nel *call stack* un metodo che contenga un blocco di codice che possa gestire l'eccezione

- ♦ La ricerca comincia nel metodo dove si verifica l'errore, e prosegue in ordine inverso rispetto alle invocazioni

1. Se c'è un gestore appropriato per il tipo di eccezione, allora il gestore *cattura* (*catch*) l'eccezione

2. Se non c'è un gestore appropriato, allora il programma termina

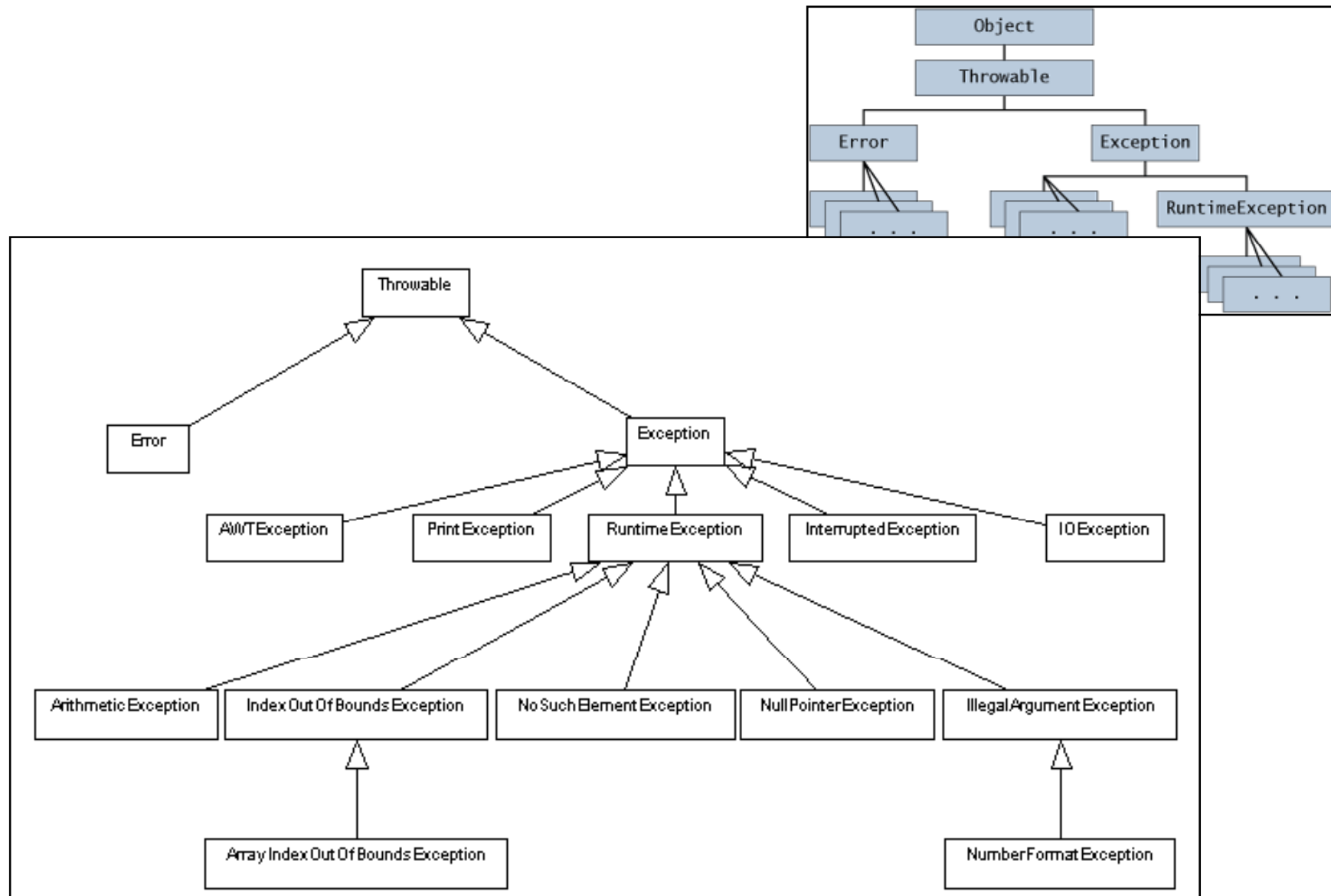


- ◆ I programmi Java, per essere compilati, devono rispettare la regola:

“Gestisci o specifica le eccezioni”

- ◆ Il codice che può lanciare un’eccezione è racchiuso:
 - In un blocco che catturi l’eccezione (handler)
 - Nel corpo di un metodo che specifichi il lancio dell’eccezione attraverso la clausola `throws`

Gerarchia di eccezioni



- ◆ Le eccezioni controllate sono delle eventi eccezionali che una applicazione ben scritta (affidabile) dovrebbe prevedere e risolvere
 - Se si passa il nome di un file per costruire un `FileReader`, ma il file non esistente
 - Allora viene lanciata una `FileNotFoundException`
- ◆ Quindi un'applicazione ben scritta deve catturare l'eccezione, notificare all'utente l'errore e chiedere un nuovo nome di file
- ◆ Le eccezioni controllate (**checked**) sono soggette alla regola “*cattura o specifica*”

- ◆ Gli errori sono eventi eccezionali esterni all'applicazione, che un'applicazione spesso non può prevedere o risolvere
 - Se si apre un file in lettura, ma la lettura non è possibile o per un guasto hardware o di sistema
 - Allora viene lanciato un `IOError`
- ◆ Visto la loro bassa probabilità e la difficoltà della gestione degli errori, è ragionevole terminare l'esecuzione senza tentare di gestirli
- ◆ Gli errori individuati da `Error` (e sottoclassi) non sono soggette alla regola “*cattura o specifica*”

- ◆ Le eccezioni di runtime sono eventi eccezionali interne che un'applicazione dovrebbe prevenire
 - Se si legge una variabile a cui non è stato precedentemente assegnato un valore
 - Allora viene lanciato una `NullPointerException`
- ◆ Quindi un'applicazione non deve catturare l'eccezione, ma il suo codice deve essere riscritto per rimuovere l'errore di programmazione
- ◆ Gli eccezioni individuate da `RuntimeException` (e sottoclassi) sono
 - ◆ Note come *eccezioni non controllate* (**unchecked**)
 - ◆ Non soggette alla regola “*cattura o specifica*”

- ◆ Un'eccezione viene catturata attraverso il costrutto:

```
try ... catch ... finally ...
```

- ◆ Il blocco seguente la parola chiave `try` contiene il codice che può lanciare una o più eccezione
- ◆ Il blocco seguente la parola chiave `catch` contiene il codice che gestisce le eccezione
- ◆ Il blocco seguente la parola chiave `finally` contiene il codice che deve essere eseguito in ogni caso (con o senza eccezioni)

```
try {  
    System.out.println("Entered try statement");  
    out = new PrintWriter(new FileWriter("OutFile.txt"));  
    for (int i = 0; i < SIZE; i++) {  
        out.println("Value at: " + i + " = "  
                    + vector.get(i));  
    }  
}  
  
catch and finally statements . . .
```

La piattaforma invoca il primo gestore nello stack il cui tipo di argomento generalizza il tipo dell'eccezione lanciata

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

Il gestore può: scrivere messaggi, risolvere il problema associato all'eccezione, e/o chiedere all'utente di prendere una decisione

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

```
public void writeList() {
    PrintWriter out = null;
    try {
        System.out.println("Entering try statement");
        out = new PrintWriter(new FileWriter("out.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + vector.get(i));
        }
    } catch (ArrayIndexOutOfBoundsException e1) {
        System.err.println("Wrong array index: " + e1.getMessage());
    } catch (IOException e2) {
        System.err.println("Caught IOException: " + e2.getMessage());
    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            try { out.close(); } catch (IOException e3) { }
        } else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

- ♦ Se un metodo non *cattura* le eccezioni controllate che possono verificarsi al suo interno, allora deve *specificare* che può lanciarle

```
public void writeList() throws IOException,  
    ArrayIndexOutOfBoundsException { . . . }
```

- ♦ Ovviamente non è obbligatorio specificare che può lanciare errori e eccezioni non controllate

```
public void writeList() throws IOException { . . . }
```

Il lancio di un'eccezione può essere forzato dal programmatore

```
public Object pop() { // in a typical Stack class
    Object obj;
    if (size == 0) {
        throw new EmptyStackException(); }
    obj = get(size - 1); set(size - 1, null); size--;
    return obj;
}
```

Una applicazione può rispondere ad una eccezione lanciandone un'altra (si dice che la prima eccezione causa la seconda)

```
try {  
    // ...  
} catch (IOException e) {  
    throw new SampleException("Other IOException", e);  
}
```


- ♦ Java non impone ai metodi di catturare o specificare gli errori eccezioni non controllate
- ♦ I programmatori possono essere tentati di scrivere codice che lancia solo eccezioni non controllate
 - Scrivere tutte le eccezioni come sottoclassi di `RuntimeException`
 - Permettono di scrivere codice senza curarsi di errori di compilazione e senza curarsi di specificare o catturare eccezioni
- ♦ Può sembrare conveniente, ma si perdono tutti i vantaggi del controllo delle eccezioni, legati alla divisione delle responsabilità

- ◆ Perché i progettisti Java hanno imposto ai metodi di specificare tutte le eccezioni lanciate al suo interno e non catturate?
- ◆ Le eccezioni lanciate da un metodo sono parte della sua interfaccia pubblica
 - Al pari dei parametri e del valore di ritorno
 - Chi chiama un metodo deve sapere quali sono le eccezioni lanciate per poter decidere cosa farne

- ♦ Una buona parte delle eccezioni sono il risultato di errori di programmazione
 - Nelle operazioni aritmetiche
 - Nei riferimenti ad oggetti e suoi metodi
 - Nell'accesso ad array e liste
- ♦ Il programma non deve risolvere o gestire questi errori
- ♦ Il programmatore deve individuare questi errori nella fase di test del programma e poi correggerli

- ◆ Trovare un bug nei programmi non è un compito facile, e nemmeno una esperienza eccitante
- ◆ Molti bug possono capitare a causa di specifiche non ben chiare o non comprese
- ◆ Il motivo della necessità di individuare i bug il prima possibile è racchiuso in una famosa frase di Barry Boehm:
 - Se trovare e correggere un problema in fase di specifica dei requisiti costa 1\$, il costo passa a 5\$ durante il progetto, \$10 durante la programmazione, \$20 durante la fase di unit test, e fino a \$200 dopo la consegna del sistema

- ◆ Se un programma può ragionevolmente gestire l'eccezione, allora l'eccezione dovrebbe essere di tipo controllato e gestita in base a
 - Una classificazione dei tipi d'errore
 - Una gestione separata delle eccezioni e(o una propagazione errori attraverso lo stack delle chiamate
- ◆ Se un programma non può far nulla per gestire l'eccezione, allora l'eccezione dovrebbe essere di tipo non controllato

```
method1(...) {  
    try {  
        ... Calls to method2 and method3  
    } catch (Exception1 e1) {  
        ...  
    } catch (Exception2 e2) {  
        ...  
    } catch (Exception3 e3) {  
        ...  
    } finally {  
        ... final common code ...  
    }  
}  
  
method2(...) throws Exception1, Exception2 { ... }  
  
method3(...) throws Exception3 { ... }
```