Fondamenti di Informatica

(L-Z)

Corso di Laurea in Ingegneria Gestionale

OOP: I/O da/su File e Java Exceptions

Prof. Stefano Mariani

Dott. Alket Cecaj

Indice

- I/O da/su file: java.io e java.nio
  - BufferedReader e FileReader
  - PrintWriter e FileWriter
  - File e percorsi
- Gestione degli errori
  - Java Exceptions
  - il costrutto try-catch e l'istruzione throws
  - Checked vs. Unchecked exceptions
- I/O revisited: il costrutto "try-with-resource"

I/O da/su FILE:
java.io e java.nio

I/O da e su file

- Finora abbiamo sempre acquisito input da tastiera
  - tramite la classe `Scanner` e i suoi metodi
- e prodotto output su schermo
  - tramite l'oggetto misterioso riferito dal campo `System.out` e i suoi metodi
- Ci sono in realtà molti mezzi di I/O possibili
  - **i files**, la rete, dispositivi specifici (e.g. stampanti e telecamere)
- Ora ci dedichiamo all'**I/O da e su file**
  - *limitatamente al poco che ci serve per l'esame*
  - l'argomento è vasto, noi vediamo i metodi più semplici (non necessariamente più efficaci) per fare I/O
- Java ci fornisce numerosi classi e metodi nei package **java.io** e **java.nio***

*Versione migliorata di java.io per molti aspetti, che non vedremo

BufferedReader e FileReader

- La classe **BufferedReader** del package java.io consente di *leggere* testo da una sorgente qualunque
  - tastiera, **file**, rete, etc.
- La classe **FileReader** del package java.io consente di *leggere* testo da **file**
  - tipicamente, un file con estensione **.txt**

```
BufferedReader br = new BufferedReader(new FileReader("src/test.txt"));
String line = br.readLine();
while (line != null) {
  System.out.println(line);
  line = br.readLine();
}
```

Esempi di metodi offerti e utilizzo nel file Slide05.java della lezione 13

PrintWriter e FileWriter

- La classe **PrintWriter** del package java.io consente di *scrivere* testo su una destinazione qualunque
  - tastiera, **file**, rete, etc.
- La classe **FileWriter** del package java.io consente di *scrivere* testo su **file**
  - tipicamente, un file con estensione **.txt**

```
BufferedReader br = new BufferedReader(new FileReader("src/test.txt"));
PrintWriter pw = new PrintWriter(new FileWriter("src/out.txt"))
String line = br.readLine();
while (line != null) {
  pw.println(line);
  line = br.readLine();
}
```

Esempi di metodi offerti e utilizzo nel file Slide06.java della lezione 13

File e Percorsi

- La classe **File** del package java.io rappresenta in astratto file e cartelle (directory) ed i relativi **percorsi**
  - *percorso assoluto* ~ sequenza di cartelle che dalla cartella radice del file system arriva al file/cartella in questione
  - *percorso relativo* ~ sequenza di cartelle che dalla cartella corrente del file system arriva al file/cartella in questione
    - *quelli che vedevate in rosso nelle slides precedenti*
- Consente di *creare/rimuovere/copiare/spostare* file e cartelle, elencare file e cartelle contenute in altre cartelle, etc.

```
File dir = new File("data/IoTests");
System.out.println(dir.getAbsolutePath());
if (!dir.exists()) {
  dir.mkdirs();
}
```

Esempi di metodi offerti e utilizzo nel file Slide07.java della lezione 13

GESTIONE ERRORI: JAVA EXCEPTIONS

Gestione errori

- Il codice degli esempi precedenti *non compila* (né esegue, ovviamente) senza aggiungere nuovi costrutti

```
try ... {
  ...
} catch (FileNotFoundException e) {
  ...
} catch (IOException e) {
  ...
}
```

- Tali costrutti sono *necessari* perché le operazioni di I/O (e non solo quelle) possono *fallire*:
  - il file non esiste, non si hanno i permessi di scrittura, etc.
- Questi errori vengono catturati da Java e segnalati al programmatore in forma di **eccezioni**…
  - *ogni* classe Java può segnalare eccezioni, non solo quelle di I/O

Java Exceptions

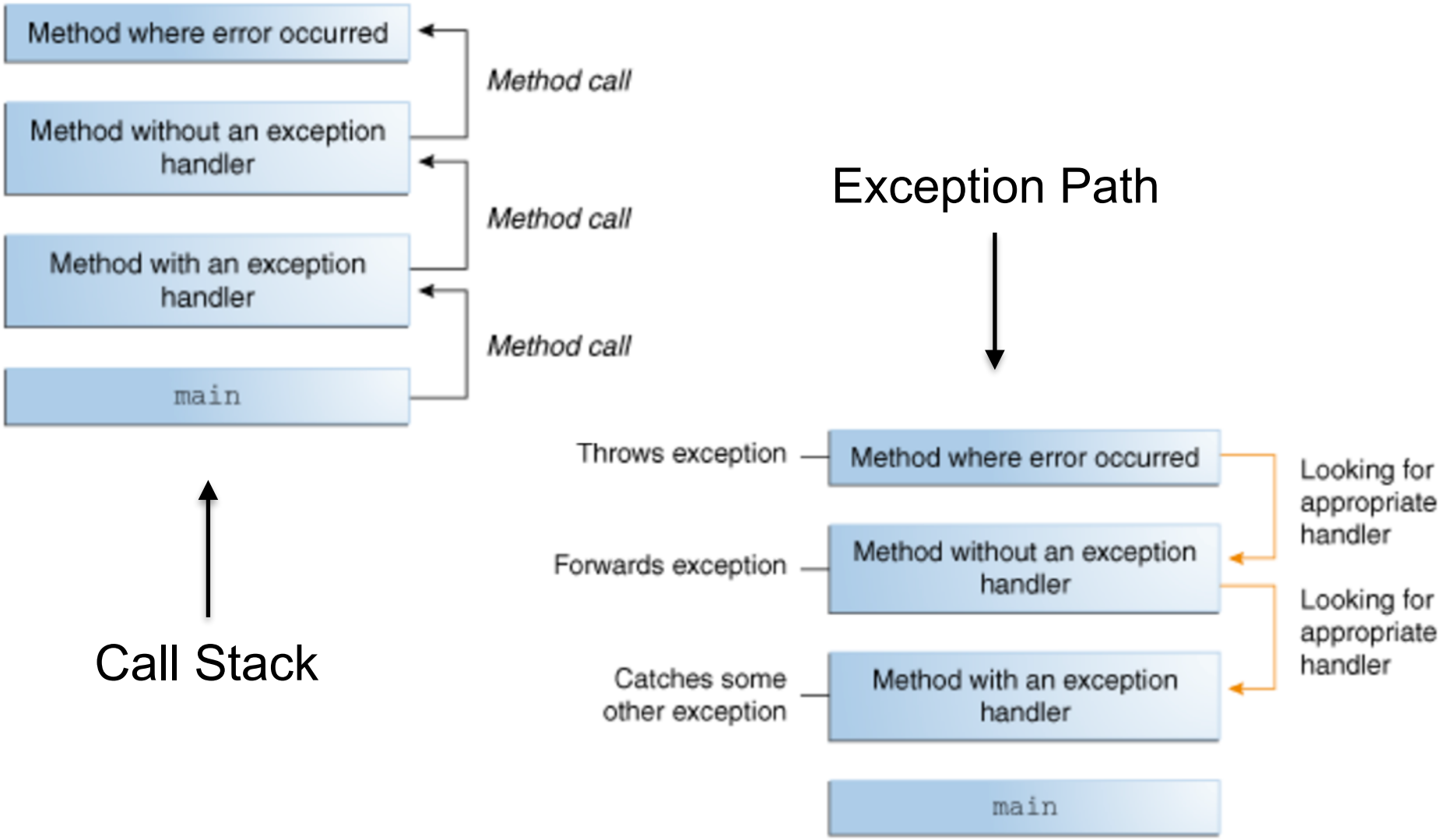
- Dalla documentazione ufficiale*:

Definition: An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

- Quando si verifica un errore, la JVM
  - crea un *oggetto* (**`Exception`**) che contiene informazioni sull'errore stesso
  - cerca *qualcuno* in grado di *gestire* l'errore
- Quel qualcuno è da cercare nella *catena di metodi* invocati per arrivare al metodo in cui si è verificato l'errore
  - detta **call stack** (~ *pila di chiamata*)

*https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

“Qualcuno”: call stack



“Gestire”: try-catch vs. throws

- Il metodo in cui si verifica l’errore crea l’eccezione, la “*lancia*” (**`throw`**) e *termina* l’esecuzione
- L’esecuzione (e l’eccezione) passa al metodo chiamante, che può:
  - *catturare* (**`catch`**) l’eccezione, gestendo la situazione anomala per *evitare* di terminare
  - lasciarla emergere (“*bubble up*”), delegando al prossimo metodo nella call stack la gestione (**`throws`**)
- Se/quando l’eccezione raggiunge il `main`, l’esecuzione del programma termina stampando un opportuno messaggio di errore (il tipo di eccezione e la *stack trace*)

“Gestire”: try-catch vs. throws

- Java fornisce ai programmatori tre costrutti/istruzioni per *gestire* le eccezioni
  - **throw** per *lanciare* un’eccezione dal *corpo* di un metodo
    - e.g. throw new IllegalArgumentException
  - **throws** per dichiarare nella *signature* di un metodo che si intende *lasciare emergere* un’eccezione
    - e.g. public void leggiDati(File f) throws IOException {...}
  - **try-catch** per catturare un’eccezione

```
File newFile = new File("data/IoTests/test.txt");
try {
  newFile.createNewFile();
} catch (IOException e) {
  e.printStackTrace();
}
```

Esempi di utilizzo nel file Slide13.java della lezione 13

Tipi di eccezioni

- Non sempre il compilatore obbliga a gestire (`try-catch` o `throws`) un'eccezione
  - e.g. `NullPointerException`: non siete *mai* stati obbligati a gestirla, ma ne avete vista la stack trace visualizzata dal main in rosso un sacco di volte :)
- Dipende dal *tipo* di eccezione
  - **Checked Exception**: eccezioni che rappresentano situazioni anomale *risolvibili*
    - e.g. `IOException`
  - **Unchecked Exception**: eccezioni che rappresentano *errori di programmazione*
    - e.g. `NullPointerException`

Il costrutto **try-with-resource**

- **Sintassi**:

```
try (<dichiarazione risorse>) {
  ... // normale flusso
} catch (<dichiarazione eccezioni>) {
  ... // situazione anomala
}
```

- E.g.:

```
try (BufferedReader br = new BufferedReader(new
FileReader("src/in.txt"))) {
  ...
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

```
try (BufferedReader br = new BufferedReader(...);
        PrintWriter pw = new PrintWriter(...)) {
  ...
} catch (FileNotFoundException | IOException e) {
  e.printStackTrace();
}
```

Fondamenti di Informatica

(L-Z)

Corso di Laurea in Ingegneria Gestionale

OOP: I/O da/su File e Java Exceptions

Prof. Stefano Mariani

Dott. Alket Cecaj