Javadoc ed Eccezioni

Laboratorio di Programmazione II

Corso di Laurea in Bioinformatica

Dipartimento di Informatica - Università di Verona

Sommario

- Lettura e scrittura su file
- Javadoc
- Eccezioni in Java

Lettura e Scrittura su file

Scrittura su file

Scrittura su file di testo

- Scrivere stringhe su file di testo
- Usiamo gli oggetti FileWriter e PrintWriter
  1. FileWriter apre un file in scrittura, ha come parametro il nome del file
  2. PrintWriter mette a disposizioni metodi di scrittura per testo (print, println), ha come parametro un oggetto FileWriter
- a scrittura terminata dobbiamo chiudere il file ed il canale di scrittura
- dobbiamo importare alcune classi del package di input-output:
  - `import java.io.*;`

Scrittura su file: Esempio

Example (Esempio scrittura su file di testo)

1. apertura del file in scrittura / creazione del file

```
FileWriter f = new FileWriter("prova.txt");
PrintWriter out = new PrintWriter(f);
```

2. scrittura di testo

```
out.println("scrivo su file");
```

3. chiusura del canale di scrittura e del file

```
out.close();
f.close();
```

Scrittura su file

Note

- FileWriter crea un file pronto per essere scritto.
- Se il file già esiste viene preliminarmente eliminato tutto il suo contenuto.
- Per aggiungere testo ad un file già esistente:
  - si usa FileWriter(String, boolean) ponendo il secondo argomento a true.
- EsempioFile.java

Schema di ciclo di scrittura

Ciclo di scrittura su file

```
PrintWriter out = ...
while (condizione) {
  out.println(dato);
  aggiorna condizione
  ...
}
out.close();
```

Esempio scrittura 10 stringhe di testo

```
FileWriter f = new FileWriter(nomefile);
PrintWriter out = new PrintWriter(f);
int i = 0;
while(i<10){
   out.println("stringa "+i);
   i++;
}
out.close();
f.close();
```

Lettura da file

Leggere stringhe da file di testo

1. Usiamo gli oggetti FileReader e BufferedReader
   - FileReader apre un file in lettura, accetta come parametro il nome del file
   - BufferedReader mette a disposizione metodi per la lettura di stringhe di testo (e.g., readLine()) accetta come parametro un oggetto FileReader
2. Leggiamo una linea alla volta con il metodo readLine() della classe BufferedReader
3. Chiudiamo il file ed il canale di input

EsempioFile.java

Lettura da file: Esempio

Example (Esempio lettura da file di testo)

1. apertura del file in lettura

```
FileReader f = new FileReader("prova.txt");
BufferedReader in = new BufferedReader(f);
```

2. lettura di una riga del file

```
String line = in.readLine();
```

3. chiusura del file

```
f.close();
in.close();
```

Schema di ciclo di lettura

Ciclo di lettura da file

```
BufferedReader in = ...
String linea = in.readLine();
while (linea!=null) {
   ...
   linea = in.readLine();
}
```

Esempio: contare le linee di un file

```
FileReader f = new FileReader(nomefile);
BufferedReader in = new BufferedReader(f);
int n = 0;
String linea = in.readLine();
while (linea!=null) {
   n++;
   linea = in.readLine();
}
f.close();
in.close();
```

Esercizi

Esercizi su scrittura e lettura

Implementare i metodi:

- `stampaStudenteFile(...)`
- `calcolaMediaStudenti(...)`
- `copiaStudentiInMedia(...)`

della classe ClienteStudenteBase.java.

Scaricare il file StudenteBase.java ed il file di testo studenti.txt per verificare il corretto funzionamento dei metodi.

Commentare il codice:Javadoc

Premessa

I commenti nel codice

- I commenti sono fondamentali per il riuso del software
- Alcune informazioni sono chiare dal codice (ad esempio l'interfaccia)
- Altre importanti informazioni sono meno chiare (ad esempio il contratto)
- Javadoc: genera la documentazione (e.g., HTML o PDF) dai commenti

Formato Javadoc

Javadoc

- Commento Javadoc: testo HTML tra i tag: /** e */

Example (Commento Javadoc)

```
/**
* Commento <em>Javadoc</em>
* Spazi ed asterischi a inizio riga
* sono <strong>sempre</strong> ignorati
*/
```

Risultato:
Commento *Javadoc*
Spazi ed asterischi a inizio riga sono **sempre** ignorati

Cosa commentare

Scopo dei commenti

- Commentare tutto ciò che può essere utile a far comprendere il codice al lettore (programmatori ed utenti)
- Documentazione minima:
  - ciascun package, classe ed interfaccia
  - ciasun metodo ed attributo pubblico

Come commentare

Scopo dei commenti

- I commenti vengono posti prima della dichiarazione del codice che si vuole commentare (classe, metodo, etc.)
- Un commento deve descrivere in modo sintetico quale è lo scopo del codice
- Un commento Javadoc può contenere tag speciali (non HTML) che aiutano ad organizzare il commento

Esempio Javadoc

Esempio

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the location of the image
 * @param  name the location of the image
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name);
```

Esempio Javadoc: discussione

Esempio: discussione

- primo paragrafo riassume lo scopo del metodo.
- paragrafi successivi sono suddivisi da <p>.
- Una riga vuota separa la parte descrittiva dalla parte dei tag. Deve esserci solo 1 riga vuota in tutto.
- Una seconda parte in cui sono inseriti tag speciali (e.g., @<nome>) che identificano le componenti dell'oggetto che si sta commentando.

Tag Javadoc

Tag Javadoc

- Formato generale di un tag: @name comment
- name specifica quale tipo di informazione si sta dando
- comment è l'informazione:
- e.g. @author William Shakespeare
- Ogni tag deve essere su una riga nuova.
- I commenti possono estendersi su più righe, ma non ci devono essere righe vuote!

Tag Javadoc

Tag Javadoc

- Ordine Tag: Tag Classi, Tag Metodi

Tag classi:

- @author <autore> il nome di chi ha scritto il programma.
  - Se ci sono più autori, si mettono più tag su righe separate
  - Appare solo se javadoc viene compilato con l'opzione –author
- @version <versione> la versione del codice
  - appare solo se javadoc viene compilato con l'opzione –version

Tag Javadoc

Tag Javadoc

Tag Metodi:

- @param <nome parametro> breve descrizione del parametro.
  - Se ci sono più parametri, rispettare l'ordine con cui sono dichiarati.
- @return breve descrizione di ciò che è tornato dal metodo.
  - Esempio: @return true if the value was found in the array
- @exception <nome eccezione> descrizione delle circostanze che determinano il lancio dell'eccezione.

Come si compila

Compilare con javadoc

- Il comando per estrarre la documentazione è javadoc presente in tutte le Java Development Kit (JDK)
- Il comando javadoc può produrre la documentazione in diversi formati, ciascuno dei quali può richiedere dei file di istruzioni speciali (i cosidetti doclet)
- Il formato standard è HTML e non richiede nessun file ausiliario.
- Il comando da lanciare è il seguente: javadoc [parametri] -d <DirectoryDestinazione> <NomeFile.java>
- Ad esempio:

```
javadoc -private -d mydocs/ Insegnamento.java
```

Informazioni aggiuntive

Ulteriori dettagli

- Se si vuole la documentazione in PDF è necessario specificare una doclet in grado di istruire Javadoc su come produrre un PDF (e.g. PdfDoclet)
- Leggere sun doc per un'esauriente spiegazione di come si devono scrivere i commenti Javadoc.
- Commentare sempre il codice con javadoc anche in sede d'esame.

Esercizi

Esercizi: commenti

- Commentare la classe Insegnamento.java in javadoc
- Generare la documentazione in html
- soluzione: Insegnamento.sol

Eccezioni

Esigenza eccezioni

Eccezioni

- Permettono di gestire condizioni al di fuori dei parametri operativi di un metodo
- Esempi tipici: errori di input, dati non disponibili, dati non validi etc.
- Tali condizioni non possono sempre essere gestite all'interno del metodo
- Eccezioni: meccanismo ben definito per gestire tali situazioni

Esempio

Example (Ricerca massimo reali da input)

- Il metodo assume che l'utente inserisca sempre almeno un dato
- Cosa succede se l'utente decide di non inserire nulla ?
-
```
Exception in thread "main"
java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:840)
at java.util.Scanner.next(Scanner.java:1461)
at java.util.Scanner.nextDouble(Scanner.java:2387)
at lezione1.MassimoReali.main(MassimoReali.java:12)
```

Definizione Eccezione

Definizione

- Oggetto creato dalla JVM a seguito di una situazione anomala
- L'oggetto contiene le informazioni riguardanti la situazione anomala che ha generato l'eccezione
- Un modulo può generare un'eccezione quando si verifica la condizione anomala (throw)
- Un modulo può gestire un'eccezione generata da qualche altro modulo (catch)
- Nella gestione di un'eccezione un modulo può generare una nuova eccezione che viene gestita da qualche altro modulo

Costrutto Java per le eccezioni

La classe Throwable

- Throwable: la superclasse per tutte le eccezioni
- Error: sottoclasse di Throwable rappresenta eccezioni che possono essere ignorate dai moduli
- Exception: sottoclasse di Throwable rappresenta eccezioni che dovrebbero essere gestite in qualche modo

Eccezioni di tipo Error

Error

- Condizioni anomale difficilmente gestibili da un modulo
- Principali sottoclassi di Error:
  - ThreadDeath
  - LinkageError
  - VirtualMachineError
- Queste eccezioni possono essere ignorate
- Quando si verificano il metodo termina segnalando il codice di errore relativo

Eccezioni di tipo Exception

Exception

Due categorie:

1. Eccezioni non controllate dal compilatore
   - Sottotipo RunTimeException
   - Il compilatore non forza la loro gestione
   - Esempi: indice fuori dai limiti di un array statico, riferimento nullo a oggetto, divisione per 0, etc.
   - Sottotipi di RunTimeException rappresentano questi casi, e.g. NullPointerException
2. Eccezioni controllate dal compilatore
   - Tutti gli altri sottotipi di Exception
   - Il compilatore forza la loro gestione
   - Il compilatore restitusce un errore di compilazione se l'eccezione viene ignorata
   - Esempio: ClienteStudenteBase.java (*leggiStudenteFile(...)*)

Utilizzo Eccezioni

Modalità di utilizzo

1. Generare Eccezioni
   - Riconoscere una situazione di errore e lanciare una eccezione.
2. Gestire Eccezioni
   - Intercettare una eccezione ed eseguire codice specifico per gestirle
3. Possibilità di gestione
   - delegare la gestione ad un altro modulo
   - intercettare (e gestire) eccezioni di uno o più tipi
   - intercettare una eccezione e generarne un'altra
   - eseguire una parte di codice indipendentemente dalla eccezione

Generare Eccezioni

Generazione

Due categorie:

1. Dopo aver individuato una condizione anomala un metodo può decidere di generare una eccezione per far gestire la situazione ad altri moduli.
2. Generazione eccezioni in Java:
   - Nella definizione del metodo dichiarare che può generare eccezioni
     - parola chiave throws nella definizione del metodo
   - Nel corpo del metodo
     - Istanziare un oggetto di tipo Exception appropriato
     - Sollevare l'eccezione creata mediante l'istruzione throw.

Esempio Generazione

Example (Esempio generazione eccezione)

```
public class MassimoGenerale {
 public static void main(String[] args)
  throws IllegalArgumentException {
   ...
   if (trovato){
    System.out.println("Massimo trovato "+max);
   }else {
    IllegalArgumentException e = new
     IllegalArgumentException("Nessun elemento inserito");
    throw e;
   }
 }
}
```

Generazione Eccezioni

Note

- NullPointerException è un sottotipo di RunTimeException quindi può non essere controllata
- throws accetta una lista di tipi di eccezioni: e.g. throws NullPointerException, IOException, ...
- L'esecuzione dell'istruzione throw causa l'interruzione del metodo, il controllo passa al metodo chiamante

Gestire Eccezioni

Intercettare eccezioni

- Se invochiamo un metodo che genera eccezioni controllate (e.g. non sottotipi di Error o RunTimeException) dobbiamo necessariamente gestire le eccezioni che possono essere generate
- 2 possibilità
  - Si dichiara che il metodo possa a sua volta sollevare eccezioni: delega
  - Si inserisce il codice necessario per risolvere l'eccezione

Delega

Delegare la gestione delle eccezioni

```
public static Studente leggiStudente(String file){
        FileReader f = new FileReader(file);
        BufferedReader in = new BufferedReader(f);
...
}
```

Questo codice non compila: Errore: "Unhandled exception type FileNotFoundException"

```
public static Studente leggiStudente(String file)
 throws FileNotFoundException{
        FileReader f = new FileReader(file);
        BufferedReader in = new BufferedReader(f);
...
}
```

Delega

Catena di delega

- Se l'eccezione delegata si verifica, il metodo termina ed il controllo passa al metodo chiamante segnalando l'eccezione
- Se anche il metodo chiamante delega la gestione di questo tipo di eccezioni si passa al successivo metodo chiamante
- La catena si interrompe quando:
  - uno dei metodi chiamanti intercetta e gestisce quel tipo di eccezioni
  - lo stack delle chiamate termina: la JVM termina l'esecuzione del programma segnalando un errore.
- Per delegare più eccezioni che appartengono alla stessa superclasse basta delegare la superclasse:
  - Per delegare: FileNotFoundException, IOException basta scrivere throws IOException

Gestire le eccezioni

Gestione diretta

Le istruzioni per gestire le eccezioni sono:

- try ... racchiude le istruzioni che possono generare le eccezioni che si vuole gestire
- catch ... racchiude il codice da eseguire quando le eccezioni si verificano
- finally ... racchiude codice che deve essere eseguito in ogni caso: sia se si è verificata un eccezione sia se non se ne è verificata nessuna.

Gestire le eccezioni: Esempio

Example (Gestione diretta)

```
public static Studente leggiStudente2(String file){
 //tolto throws
  FileReader f = null;
  try {
   //codice che può lanciare una o più eccezioni
   f = new FileReader(file);
  } catch (FileNotFoundException e) {
   //codice che gestisce l'eccezione di
   tipo FileNotFoundException
   System.out.println(
    "Il file" + file + "non è stato trovato");
  }
  ...
}
```

Gestire le eccezioni: note

Gestione diretta: note

- il blocco try viene eseguito normalmente
- se si verifica un eccezione nel blocco try
  - si interrompe l'esecuzione
  - se esiste un blocco catch corrispondente si eseguono le istruzioni nel blocco catch e poi si prosegue con il codice dopo il blocco catch.
  - se non esiste un blocco catch (eccezioni Error o RunTimeException) si interrompe il metodo e si passa al chiamante
- se non si verifica un eccezione nel blocco try il blocco catch non viene eseguito, si prosegue con il codice seguente.

Gestire più di un eccezione: Esempio

Example (Gestione diretta: più eccezioni)

```
public static Studente leggiStudente3(String file){
 //tolto throws
  FileReader f = null;
  BufferedReader in = null;
  String s = null;
  try {
  //codice che può lanciare una o più istruzioni
   f = new FileReader(file);
   in = new BufferedReader(f);
   s = in.readLine();
  } catch (FileNotFoundException e1) {
   //codice che gestisce l'eccezione FileNotFoundException
   System.out.println(file + " non trovato");
  } catch (IOException e2) {
   //codice che gestisce l'eccezione IOException
   System.out.println("Problema nella lettura da file");
}
```

Gestire le eccezioni: note

Gestione diretta: note

- Viene eseguito il blocco catch corrispondente al tipo di eccezione intercettata
- Le eccezioni gestite debbono essere di tipi differenti, altrimenti il compilatore restituisce un errore (Unreachable catch block ...)
- I catch vengono valutati in sequenza quindi le sottoclassi debbono stare più in alto delle superclassi:
  - nell'esempio precedente
  - Il blocco IOException non può stare prima di FileNotFoundException

Gestire le eccezioni

Note

- Gestire un'eccezione vuol dire fare delle operazioni particolari se l'eccezione si verifica o cambiare il flusso del codice in presenza dell'eccezione
- Il semplice catch dell'istruzione può modificare il flusso di esecuzione
- Esempio: metodo esempioGestione in Test.java

Gestire le eccezioni: **finally**

Gestione diretta: il blocco finally

- Utile per eseguire istruzioni a prescindere dalle eccezioni
- Tipicamente usato per rilasciare risorse (e.g., chiudere un file)
- Esempio: metodo esempioFinally in Test.java

Esercizi I

Esercizi I

- Realizzare il metodo calcolaMedia() della classe Test.java. Il metodo deve leggere 3 interi da input, mediante un ciclo while che termina quando l'utente inserisce un dato non intero. Se l'utente inserisce esattamente 3 interi il metodo deve calcolarne la media. Altrimenti il metodo deve generare una IllegalArgumentException
- Realizzare il metodo confrontaMedia() della classe Test.java, per calcolare due medie. Il metodo utilizza calcolaMedia(), per calcolare due medie e stampa la media più alta. Il metodo deve gestire le eccezioni sollevate da calcolaMedia().

Esercizi II

Esercizi II

- Realizzare il metodo calcolaMediaFile(String file) della classe Test.java. Il metodo opera come calcola media ma legge i dati da file. Il metodo deve delegare la gestione di tutte le eccezioni.
- Realizzare il metodo confrontaMediaFile(String file1, String file2) della classe Test.java. Il metodo utilizza calcolaMediaFile(), per calcolare due medie e stampa la media più alta. Il metodo deve gestire le eccezioni solevate da calcolaMediaFile()
  Nota: per eseguire il della classe Test.java scaricare i seguenti file txt:
  - media1.txt
  - media2.txt
  - input.txt