Programmazione Orientata agli Oggetti in Linguaggio Java

Eccezioni: Conclusioni

versione 2.0

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Riepilogo
- Evoluzione del Linguaggio
- Eccezioni
  - ⇨Trasformare le Eccezioni
  - ⇨Definire Nuove Eccezioni

Riepilogo

- Qualità del codice
  - ⇨è un obiettivo fondamentale di questo corso
  - ⇨corretta impostazione metodologica
- Le qualità da perseguire
  - ⇨correttezza
  - ⇨usabilità
  - ⇨manutenibilità
  - ⇨efficienza

Riepilogo

ATTENZIONE
a come cambia
il processo di sviluppo

- Il processo di sviluppo discusso finora
  - ⇨è centrato sulla corretta attribuzione delle responsabilità che garantisce buona organizzazione
- La nuova linea guida
  - ⇨adottare uno stile di programmazione difensiva

Riepilogo

- Programmazione difensiva
  - ⇨per ciascun metodo, valutare le pre-condizioni e in caso di possibili condizioni eccezionali lanciare le corrispondenti eccezioni
  - ⇨negli schermi, effettuare sistematicamente la convalida dei dati dell'utente e le verifiche necessarie prima di chiamare i metodi

Evoluzione del Linguaggio

- In questo modulo
  - ⇨una nuova parola chiave del linguaggio: assert
  - ⇨in precedenza, la parola chiave enum
  - ⇨si tratta di forme di evoluzione del linguaggio
  - ⇨entrambe richiedono di utilizzare l'opzione –source del compilatore
- Di seguito
  - ⇨riassumiamo le principali evoluzioni viste

Evoluzione del Linguaggio

- J2SE 1.2
  - ⇨notevoli modifiche nei package fondamentali (es: java.util, java.io) rispetto a Java 1.1
  - ⇨ma non cambia significativamente la sintassi rispetto a quella di Java 1.1
- J2SE 1.3
  - ⇨sostanzialmente identico a J2SE 1.2
  - ⇨cresce il numero di classi e di package delle API

Evoluzione del Linguaggio

- J2SE 1.4
  - ⇨alcune modifiche importanti nei package fondamentali (es: java.nio, gestione di XML: JAXP)
  - ⇨nuova funzionalità: asserzioni; parola chiave assert
- Nuovo livello dei sorgenti
  - ⇨l'opzione -source 1.4 abilita le asserzioni

Evoluzione del Linguaggio

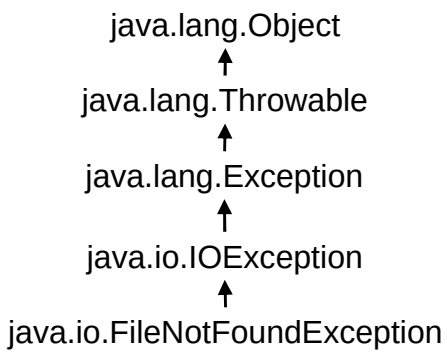
- J2SE 1.5 – 5.0
  - ⇨numerosi cambiamenti nelle API
  - ⇨numerosi cambiamenti significativi nel linguaggio (tipi generici, boxing ed unboxing automatico, ...)
  - ⇨enumerazioni; nuova parola chiave enum
- Nuovo livello dei sorgenti
  - ⇨l'opzione –source 1.5 abilita le enumerazioni
  - ⇨nota: le asserzioni sono sempre abilitate

Eccezioni

- Nei metodi per la persistenza
  - ⇨è necessario lavorare con varie eccezioni
  - ⇨controllate e non controllate
- Eccezioni controllate
  - ⇨estendono java.lang.Exception
- Eccezioni non controllate
  - ⇨estendono java.lang.RuntimeException
- Esistono ulteriori ramificazioni

Eccezioni

- Esempio: java.io.FileNotFoundException



Eccezioni

- Attenzione all'uso della clausola catch
  - ⇨il polimorfismo interferisce con la cattura delle eccezioni
- La regola
  - ⇨un'eccezione viene catturata dal primo blocco catch il cui riferimento è compatibile con il tipo dell'eccezione
  - ⇨l'eccezione potrebbe essere compatibile con più clausole catch

Eccezioni

- Esempio
  - ⇨caricamento dei dati dello studente
- Una prima versione standard
  - ⇨crea un BufferedReader da un FileReader
  - ⇨cattura le eccezioni e le stampa sullo schermo

```
public static Studente carica(String nomeFile) {
      Studente studente = new Studente();
      java.io.BufferedReader flusso = null;
      try {
         java.io.FileReader fileReader = new java.io.FileReader(nomeFile);
         flusso = new java.io.BufferedReader(fileReader);
         estraiDatiStudente(studente, flusso);
      } catch (java.io.FileNotFoundException fnfe) {
         System.out.println(fnfe);
      } catch (java.io.IOException ioe) {
         System.out.println(ioe);
      } finally {
         try {
            if (flusso != null) {
               flusso.close();
            }
         } catch (java.io.IOException ioe) {}
      }
      return studente;
   }
```

Eccezioni

- Alcune annotazioni
  - ⇨i due blocchi catch devono essere esattamente nell'ordine specificato, altrimenti il secondo diventa inutile
  - ⇨FileNotFoundException viene intercettato dal primo blocco
  - ⇨ma la soluzione vista è decisamente migliorabile

Eccezioni

- Primo miglioramento
  - ⇨non vengono effettivamente catturate tutte le eccezioni che possono verificarsi durante il caricamento (es: NumberFormatException, NullPointerException)
  - ⇨I alternativa: specificare un blocco catch per ciascuna
  - ⇨II alternativa: considerarle tutte equivalenti se non per il messaggio che portano e per la conseguenza: non è possibile caricare il file

Eccezioni

- In questo caso
  - ⇨la seconda alternativa sembra migliore
  - ⇨posso pensare di catturare tutte le possibili eccezioni indistintamente con un blocco del tipo catch (Exception e) {...}

```
try {
    java.io.FileReader fileReader = new java.io.FileReader(nomeFile);
    flusso = new java.io.BufferedReader(fileReader);
    estraiDatiStudente(studente, flusso);
} catch (Exception e) { ...
} finally { ...
}
```


Eccezioni

- Attenzione
  - ⇨questo vale solo se le diverse eccezioni devono essere gestite allo stesso modo
  - ⇨e se sono molte
  - ⇨il vantaggio è che il codice si snellisce (si evitano molti blocchi catch)
  - ⇨ma lo svantaggio è che diventa meno leggibile e c'è meno controllo sul tipo di eccezioni che viaggiano

Eccezioni

- Secondo miglioramento
  - ⇨non sembra particolarmente efficace stampare l'eccezione in quel punto
  - ⇨l'eccezione dovrebbe essere più appropriatamente gestita dal controllo
  - ⇨visto che si tratta di un evento decisamente importante per il prosieguo dell'applicazione
  - ⇨il metodo potrebbe farsi attraversare dall'eccezione

Eccezioni

- Terzo miglioramento
  - ⇨sarebbe opportuno trasformare l'eccezione prima di rilanciarla al controllo
  - ⇨in modo che il controllo non debba preoccuparsi di dettagli relativi ai file
  - ⇨ma sappia semplicemente che c'è stato un problema sulla persistenza
  - ⇨utile per disaccoppiare il controllo dalla tecnologia specifica della persistenza

Trasformare le Eccezioni

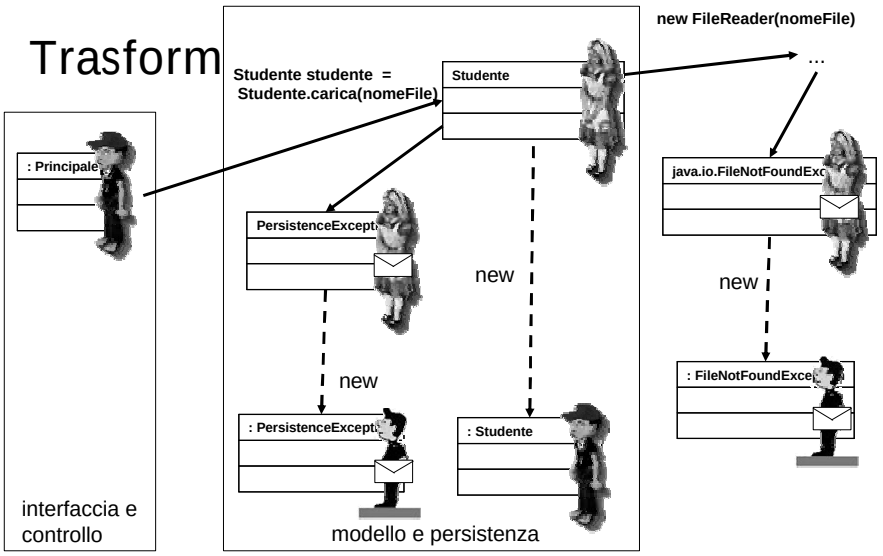
- In concreto
  - ⇨questo è fattibile dichiarando una nuova classe di eccezione
  - ⇨PersistenceException
  - ⇨in it.unibas.mediapesata.modello
  - ⇨un semplice “involucro” per incapsulare eccezioni diverse verificatesi durante il caricamento

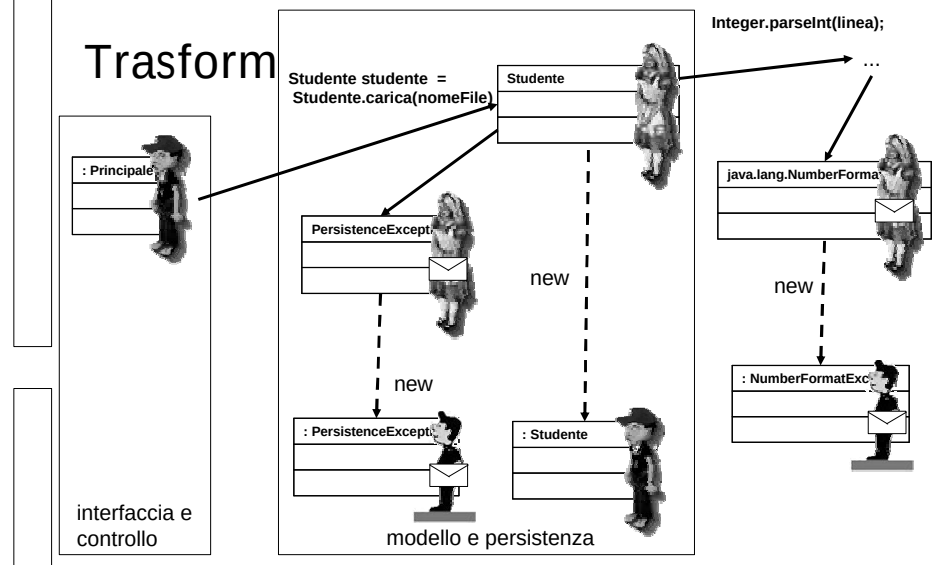
```
public static Studente carica(String nomeFile) throws PersistenceException {
    Studente studente = new Studente();
    java.io.BufferedReader flusso = null;
    try {
        java.io.FileReader fileReader = new java.io.FileReader(nomeFile);
        flusso = new java.io.BufferedReader(fileReader);
        estraiDatiStudente(studente, flusso);
        caricaEsami(studente, flusso);
    } catch (Exception e) {
        throw new PersistenceException(e);
    } finally {
        try {
            if (flusso != null) {
                flusso.close();
            }
        } catch (java.io.IOException ioe) {}
    }
    return studente;
}
```

```
package it.unibas.mediapesata.controllo;

public class Principale {

 private Studente schermoCaricaDati() {
        System.out.println("---------------------------------");
        System.out.println("       Caricamento Dati");
        System.out.println("---------------------------------");
        System.out.print("Inserisci il nome del file --> ");
        String nomeFile = it.unibas.utilita.Console.leggiStringa();
        Studente studente = null;
        try {
           studente = Studente.carica(nomeFile);
           System.out.println(" -- Caricamento effettuato -- ");
        } catch (PersistenceException ioe) {
           System.out.println("ERRORE: " + ioe);
        }
        return studente;
    }
}
```



Definire Nuove Eccezioni

- La definizione di PersistenceException
  - ⇨si tratta di un nuovo tipo di eccezione non previsto dalle API di Java
  - ⇨scelgo che sia un'eccezione controllata
  - ⇨estende java.lang.Exception
  - ⇨eredita la proprietà message
  - ⇨è necessario definire esclusivamente i costruttori

```
package it.unibas.appuntamenti.modello;

public class PersistenceException extends Exception {

    public PersistenceException() {
        super();
    }

    public PersistenceException(String s) {
        super(s);
    }

    public PersistenceException(Exception e) {
        super(e);
    }

}
```


Riassumendo

- Riepilogo
- Evoluzione del Linguaggio
- Eccezioni
  - ⇨Trasformare le Eccezioni
  - ⇨Definire Nuove Eccezioni

Termini della Licenza