Programmazione Orientata agli Oggetti in Linguaggio Java

Eccezioni: Programmazione Difensiva

versione 2.4

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

Sommario

- Tipi di Eccezioni
- Stile di Programmazione Difensiva
- Asserzioni
- Quando Usare le Accezioni
- Linee Guida

Tipi di Eccezioni

ATTENZIONE
alla classificazione
delle eccezioni

- Tre tipi di eccezioni
  - errori – estendono java.lang.Throwable
  - eccezioni controllate – estendono java.lang.Exception e java.lang.Throwable
  - eccezioni non controllate – estendono java.lang.RuntimeException e java.lang.Exception e java.lang.Throwable

Tipi di Eccezioni

- Nel seguito
  - criteri metodologici per classificare ed usare i vari tipi di eccezioni
- Errori
  - semplici metodologicamente da capire
  - sono tipicamente considerate gravi malfunzionamenti
  - sono irrecuperabili e non devono essere gestiti

Tipi di Eccezioni

- Eccezione controllata
  - ⇨anche queste sono abbastanza semplici da capire e da utilizzare
  - ⇨eccezione che, a giudizio del programmatore, richiede un intervento da parte del chiamante nel caso si verifichi
  - ⇨molto spesso recuperabile (anche se a volte possono essere irrecuperabili)

Tipi di Eccezioni

- Sistema esterno
  - ⇨sistema software al di fuori dello strato del modello e del controllo dell'applicazione
- Filosofia
  - ⇨modello e controllo dovrebbero essere "irrobustite" per fare fronte a malfunzionamenti in altre parti dell'applicaz.
  - ⇨anche a costo di un certo appesantimento del codice del chiamante

Tipi di Eccezioni

- Eccezione non controllata
  - è il tipo meno semplice da comprendere
  - eccezione che, a giudizio del programmatore, NON dovrebbe verificarsi
  - come detto, tipicamente riguarda un malfunzionamento interno al controllo o al modello
  - ovvero sarebbe evitabile con maggiore cura nella programmazione o da parte dell'utente

Tipi di Eccezioni

- Caratteristiche tipiche di queste eccezioni
  - l'eccezione non controllata non appesantisce il codice del chiamante perchè tipicamente non viene catturata
  - d'altro canto, se l'eccezione si verifica, l'applicazione abortisce
  - come visto, esistono eccezioni a questa regola (es: NumberFormatException – tipicamente viene catturata)

Tipi di Eccezioni

ATTENZIONE all'uso delle eccezioni non controllate

- Funzione dell'eccezione non controllata
  - ⇨è sostanzialmente uno strumento per il debugging durante lo sviluppo del codice
  - ⇨utile se si verifica durante i test
  - ⇨in questo caso il fallimento dell'applicazione aiuta il programmatore ad individuare un errore logico da eliminare

Stile di Programmazione Difensiva

- La linea guida fondamentale
  - ⇨adottare uno stile di programmazione "difensiva" utilizzando estensivamente le eccezioni
- Programmazione difensiva
  - ⇨due funzioni
  - ⇨irrobustire il codice
  - ⇨migliorare il processo di debugging

Stile di Programmazione Difensiva

- Programmazione difensiva
  - richiede di adottare due atteggiamenti
  - atteggiamento difensivo nella chiamata dei metodi per “prevenire” l’eccezione
  - atteggiamento difensivo nel codice dei metodi per segnalare utilizzi scorretti
- Idea
  - garantire che i metodi vengano eseguiti nelle condizioni corrette

Stile di Programmazione Difensiva

- Programmazione difensiva nella chiamata
  - prima di chiamare un metodo, è indispensabile effettuare tutte le convalide necessarie per garantire che il metodo venga eseguito su dati corretti
  - caso tipico: dei dati forniti dall’utente
  - questo dovrebbe prevenire la maggior parte delle eccezioni non controllate

Esempio: Gestisci Tentativo

istruzione di acquisizione

```
private int schermoLeggiTentativo(Partita partita) {
      System.out.println("\n--------------------------------");
      System.out.println("\n***** " + partita.getSuggerimento() + " *****");
      System.out.println("\nFinora hai effettuato " +
      partita.getNumeroDiTentativi() + " tentativi");
      System.out.print("\n--> " + partita.getNome());
      System.out.prinl(", inserisci il tuo tentativo (0 per smettere): ");
      int tentativo = it.unibas.utilita.Console.leggiIntero();
      while ((tentativo < 0) || (tentativo > 100)) {
         System.out.println("    Errore. Il valore deve essere tra 1 e 100");
         tentativo = it.unibas.utilita.Console.leggiIntero();
      }
      return tentativo;
   }
```

ciclo “standard” di convalida

Stile di Programmazione Difensiva

- Nota sulle convalide
  - ⇨ le verifiche e le convalide devono però essere ragionevolmente semplici
  - ⇨ altrimenti il codice del metodo chiamante si appesantisce troppo
- Nel caso di verifiche complesse
  - ⇨ ha senso provare ad effettuare la chiamata e lasciare al metodo il compito di effettuare la verifica
  - ⇨ preparandosi a catturare l’eventuale eccezione
  - ⇨ es: NumberFormatException, FileNotFoundException

Stile di Programmazione Difensiva

- Programmazione difensiva nel metodo
  - ⇨prima di eseguire il codice del metodo bisogna verificare che il metodo sia eseguito in condizioni corrette
  - ⇨ovvero che siano soddisfatte le precondizioni opportune
  - ⇨richiede di effettuare le verifiche e lanciare eccezioni nel caso le verifiche falliscano
  - ⇨le eccezioni sono spesso NON controllate

Esempio: gestisciTentativo

istruzione di verifica di una precondizione o “guardia”

```
public void gestisciTentativo(int tentativo) {
    if ((tentativo < 1) || (tentativo > 100)) {
        throw new IllegalArgumentException(“Valore scorretto");
    }
    this.numeroDiTentativi++;
    if (tentativo == this.numeroDaIndovinare) {
        this.trovato = true;
        this.suggerimento = "Numero indovinato";
    } else if (tentativo < numeroDaIndovinare) {
        this.suggerimento = "Prova con un numero piu' alto";
    } else if (tentativo > numeroDaIndovinare) {
        this.suggerimento = "Prova con un numero piu' basso";
    }
}
```

Esempio: checkRecord

altro esempio di “guardia”

```
public void checkRecord(int tentativi) {
    if (tentativi < 1) {
        throw new IllegalArgumentException("Numero scorretto");
    }
    this.nuovoRecord = false;
    this.uguagliato = false;
    int valoreRecord = getValoreRecord();
    if (tentativi < valoreRecord) {
        setValoreRecord(tentativi);
        this.nuovoRecord = true;
    } else if (tentativi == valoreRecord) {
        this.uguagliato = true;
    }
}
```


Stile di Programmazione Difensiva

- Guardia
  - ⇨istruzione che verifica una precondizione per l’esecuzione corretta di un metodo
  - ⇨nel caso in cui la precondizione non sia verificata lancia un’eccezione
- Dove disporre le guardie
  - ⇨una buona pratica è disporre tutte le guardie all’inizio del metodo, prima delle vere istruzioni

Stile di Programmazione Difensiva

- Utilizzare le guardie
  - ⇨impedisce di utilizzare i componenti in modo scorretto
  - ⇨in altri termini: fornisce un promemoria al programmatore durante lo sviluppo nel caso ometta o eviti le convalide nel controllo
  - ⇨questo è importante perchè un componente può essere utilizzato anche in applicazioni diverse da quella attuale

Asserzioni

- La forma più semplice di guardia
  - ⇨istruzione if – then
- Due svantaggi
  - ⇨gli if sono istruzioni “ingombranti” – molte guardie tendono ad allungare i metodi
  - ⇨gli if restano nel codice anche dopo la fase di sviluppo, e rallentano leggermente l’esecuzione dell’applicazione

Asserzioni

- Un esempio
  - costruttore di una classe Data

```
public Data (int gg, int mese, int anno) {
      if (gg < 1 || gg > 31) {
         throw new IllegalArgumentException("Giorno scorretto: " + gg);
      }
      if (mese < 1 || mese > 12) {
         throw new IllegalArgumentException("Mese scorretto: " + mese);
      }
      if (anno < 0) {
         throw new IllegalArgumentException("Anno scorretto: " + anno);
      }
      ...
}
```


Asserzioni

- In effetti
  - le guardie servono quasi esclusivamente nella fase di sviluppo e correzione
- Una soluzione migliore
  - sarebbe quella di utilizzare delle istruzioni che fossero più compatte degli if
  - e che potessero essere abilitate e disabilitate in modo da essere escluse dalle versioni in produzione

Asserzioni

- La soluzione
  - ⇨utilizzare istruzioni per effettuare asserzioni
- Asserzione
  - ⇨espressione a valori booleani che ci sia aspetta che sia vera in un certo punto del programma
  - ⇨normalmente ha associata una stringa che descrive l'asserzione

Asserzioni

- L'istruzione assert
  - ⇨serve ad effettuare un'asserzione
  - ⇨se l'asserzione è vera non produce effetti
  - ⇨se l'asserzione è falsa solleva AssertionError, un'eccezione di tipo errore, che contiene il messaggio specificato
- Sintassi
  - ⇨assert (<espressione>) : "<messaggio>"

Esempi di Asserzioni

```
public void checkRecord(int tentativi) {
    assert(tentativi >= 1) : "Numero scorretto: " + tentativi;
    ...
}
public void gestisciTentativo(int tentativo) {
    assert(tentativo >= 1 && tentativo <= 100) :
                            "Numero scorretto: " + tentativo;
    ...
}
public Data (int gg, int mese, int anno) {
    assert(gg >= 1 && gg <= 31) : "Giorno scorretto" + gg;
    assert(mese >= 1 && mese <= 12) : "Mese scorretto" + mese;
    assert(anno > 0) : "Anno scorretto" + anno;
    ...
}
```


Asserzioni

- L'istruzione assert
  - ⇨parola chiave introdotta in J2SE 1.4
  - ⇨deve essere segnalata al compilatore con l'opzione –source 1.4 di javac
  - ⇨con jdk1.5 le asserzioni sono abilitate per default e non è necessario specificare opzioni in fase di compilazione (ma può essere utilizzata l'opzione –source 1.4)

Asserzioni

- La caratteristica più interessante di assert
  - ⇨la macchina virtuale può abilitare o disabilitare le asserzioni
  - ⇨normalmente le asserzioni sono disabilitate e non producono alcun effetto
  - ⇨per abilitarle durante la fase di sviluppo è necessario utilizzare l'opzione –enableassertions oppure -ea

Asserzioni

- In sintesi
  - ⇨utilizzando il compilatore javac versione 1.5, il codice delle asserzioni viene compilato e incluso nel bytecode dell'applicazione
  - ⇨eseguendo l'applicazione con java versione 1.5, le asserzioni sono disabilitate e non vengono verificate
  - ⇨eseguendo l'applicazione con java –ea le asserzioni vengono abilitate

Asserzioni

>> morracinese

- Un esempio
  - la morraCinese
  - sviluppata con uno stile di programmazione difensiva
  - le guardie sono scritte utilizzando le asserzioni

Asserzioni

- Riassumendo
  - durante il ciclo di sviluppo il codice con le asserzioni deve essere compilato con il compilatore versione 1.5 (o con l'opzione –source 1.4)
  - e il codice viene eseguito con l'opzione –ea per abilitare le asserzioni
  - durante la fase di produzione, il codice viene eseguito senza l'opzione –ea per disabilitare le eccezioni

Linee Guida

- Linea guida n. 1
  - ⇨utilizzare uno stile di programmazione difensiva
- Linea guida n. 2
  - ⇨quando lanciare un'eccezione in un metodo?
  - ⇨solo in caso di condizioni "critiche"
  - ⇨in altri termini, non è opportuno lanciare eccezioni se il metodo può regolarmente concludersi

Linee Guida

- Infatti
  - ⇨le eccezioni non eliminano del tutto i valori speciali di ritorno
- Esempio
  - ⇨ricerca di un oggetto in una collezione (array o lista)
  - ⇨è pratica comune restituire il riferimento se l'oggetto viene trovato, oppure null
  - ⇨non è necessario lanciare un'eccezione

Linee Guida

- Altro esempio
  - ricerca della posizione di un carattere in una stringa
  - metodo public int indexOf(char c) di String
  - se il carattere non è presente, viene restituito il valore -1
  - es: int indice = “prova”.indexOf(‘v’); // 3
  - es: int indice = “prova”.indexOf(‘x’); // -1

Linee Guida

- Linea guida n. 3
  - evitare blocchi catch vuoti
  - un blocco catch vuoto nasconde l’eccezione
  - specificare sempre almeno una istruzione di stampa
- Esempio

```
try { ... } catch (IOException e) {
    System.out.println(“Eccezione:“ + e);
}
```

Linee Guida

- Linee guida n. 4
  - ⇨non sempre le eccezioni possono essere gestite
  - ⇨distinguere le eccezioni “recuperabili” >> l’applicazione può andare avanti catturandole
  - ⇨da quelle irrecuperabili (condizioni troppo gravi) >> l’applicazione deve abortire

Linee Guida

- Esempio
  - ⇨IOException nel caso di Indovina il Numero è una situazione gestibile: il gioco può continuare
  - ⇨in altri casi potrebbe essere troppo grave per proseguire e di fatto rendere il programma ingestibile
  - ⇨esempio: un programma che gestisce un archivio di contatti memorizzato su disco
  - ⇨in questi casi è opportuno comunque gestire l’eccezione e stampare le cause di errore, ma poi c’è poco altro da fare (l’utente è costretto a uscire)

Linee Guida

- Linea guida n. 5
  - ⇨gestire il ciclo di vita delle risorse esterne
  - ⇨garantendone il rilascio nel blocco finally
  - ⇨i metodi che creano flussi o connessioni verso sistemi esterni devono rilasciare questi flussi al termine dell'esecuzione, riportando le corrispondenti risorse in uno stato corretto

Riassumendo

- Tipi di Eccezioni
- Stile di Programmazione Difensiva
- Asserzioni
- Linee Guida

Quando Usare le Asserzioni

ATTENZIONE
al criterio per usare le asserzioni

- Attenzione
  - ⇨il fallimento di un'asserzione provoca praticamente sempre l'interruzione dell'applicazione (errore)
  - ⇨inoltre le asserzioni sono disabilitate in produzione
  - ⇨di conseguenza le asserzioni possono essere usate per quelle guardie per le quali non ci si aspetta che l'eccezione generata venga catturata

Quando Usare le Asserzioni

- In altri casi
  - ⇨potrebbe essere opportuno utilizzare la forma più tradizionale di guardia basata su if
  - ⇨per consentire di lanciare un altro tipo di eccezione (ed eventualmente farla catturare)
  - ⇨e lasciare la guardia in funzione anche durante la fase di produzione

Quando Usare le Asserzioni

- Sviluppo di un metodo
  - ⇨due possibili casi
- I caso: uso completamente predicibile
  - ⇨il metodo è fatto per essere chiamato solo all'interno di una certa applicazione
- II caso: uso non predicibile
  - ⇨il metodo fa parte di una libreria e sarà utilizzato da altri sviluppatori

Quando Usare le Asserzioni

- Linea guida
  - ⇨nel I caso è opportuno sviluppare le guardie usando le asserzioni
  - ⇨nel II caso è opportuno utilizzare gli if
- Esempio
  - ⇨il costruttore di Data
  - ⇨se la classe appartiene ad una libreria è opportuno mantenere gli if

Termini della Licenza