Gestione delle Anomalie

Programmazione

Corso di Laurea in Informatica
Corso di Laurea in Informatica per la Comunicazione Digitale

Università di Camerino

Prof. Michele Loreti

Anno Accademico 2024/25

Gestione (mancata) delle anomalie

```
class Recipiente {
    ...
    public void aggiungi(int quantita) {
        // 0 <= quantita
        contenuto = Math.min(contenuto + quantita, volume);
    }
    ...
}
```

- il commento documenta le pre-condizioni che devono essere rispettate per l'invocazione di `aggiungi`
- questa realizzazione di `aggiungi` **non è robusta**, se si invoca il metodo con una quantità negativa il codice viene eseguito (e l'invariante di classe può essere violato)

Versione (apparentemente) robusta di `aggiungi`

```
class Recipiente {
    ...
    public boolean aggiungi(int quantita) {
        if (quantita < 0) return false;
        contenuto = Math.min(contenuto + quantita, volume);
        return true;
    }
    ...
}
```

- questa versione di `aggiungi` comunica l'esito dell'operazione, il valore `false` segnala che l'operazione non è andata a buon fine
- **problema 1**: la responsabilità di controllare che l'operazione sia andata a buon fine resta sul **chiamante**, che può dimenticarsene facilmente
- **problema 2**: non sempre si può "sacrificare" un valore del tipo di ritorno (es. `false`) per segnalare eventuali problemi

Versione di `aggiungi` con eccezione

```
class Recipiente {
    ...
    public void aggiungi(int quantita) {
        if (quantita < 0)
            throw new IllegalArgumentException("quantita non valida");
        contenuto = Math.min(contenuto + quantita, volume);
    }
    ...
}
```

- questa versione di `aggiungi` **lancia un'eccezione** di tipo `IllegalArgumentException` per segnalare l'anomalia
- le eccezioni sono **oggetti** creati con new, la libreria standard di Java contiene numerose classi che rappresentano **tipi diversi di anomalie**
- il lancio dell'eccezione avviene con il comando `throw` (parola chiave) e termina l'esecuzione del metodo. Se non gestita, l'eccezione termina il chiamante e così via fino alla terminazione dell'intero programma
- non occorre sacrificare alcun valore del tipo di ritorno

Gerarchia di eccezioni: anomalie catastrofiche

Command Not Found: mmdc

Sotto classi di `Error`

- `OutOfMemoryError`
  (memoria heap esaurita)
- `StackOverflowError`
  (pila dei frame piena)
- …

In generale si tratta di anomalie **catastrofiche** che non ha senso cercare di rimediare, e a volte non è fisicamente possibile (se è finita la memoria c'è poco da fare…)

Gerarchia di eccezioni: anomalie interne

Command Not Found: mmdc

Sotto classi di `RuntimeException`

- `IllegalArgumentException` (`Integer.parseInt("ciao")`)
- `ArithmeticException` (divisione per zero)
- `IndexOutOfBoundsException` (accesso ad array con indice non valido)
- …

In generale si tratta di anomalie la cui origine è **interna** al programma e che ha poco senso cercare di rimediare (se è stato invocato un metodo passando un argomento illegale significa che c'è un errore logico nel programma…)

Nota: **tutte** le anomalie si presentano durante l'esecuzione del programma (ovvero "**a runtime**"), quindi il nome `RuntimeException` non è particolarmente esplicativo né distintivo di questa famiglia di eccezioni

Gerarchia di eccezioni: anomalie esterne

Command Not Found: mmdc

Sotto classi di `Exception` ma non di `RuntimeException`

- `IOException`
  (errore di input/output)
- `TimeoutException`
  (tempo scaduto per completamento operazione)
- …

In generale si tratta di anomalie che possono verificarsi durante l'esecuzione del programma ma la cui origine è **esterna** al programma

Tali anomalie **devono** essere gestite (es. è "normale" dover gestire il caso in cui una connessione a un server remoto cade o non viene stabilita)

Il compilatore Java **impone** che tutte le eccezioni sotto classe di `Exception` ma non di `RuntimeException` siano gestite

Lancio di una eccezione

Il comando per il lancio di una eccezione ha la forma

```
throw e;
```

dove e è una espressione di tipo `Throwable` (o sotto tipo di `Throwable`)

In molti casi, l'eccezione da lanciare viene creata "al volo":

```
throw new IllegalArgumentException("quantita non valida");
```

Il lancio di una eccezione causa la terminazione anomala del metodo in cui avviene il lancio e, a ritroso, di tutti i metodi in esecuzione fino a quando

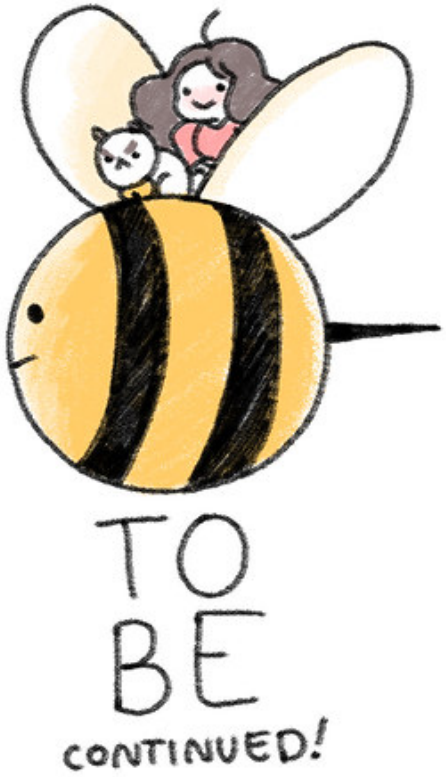
- non c'è un metodo che è in grado di **catturare** e gestire l'eccezione, oppure
- tutti i metodi in esecuzione vengono terminati, nel qual caso l'intero programma termina

Cattura e gestione di una eccezione

```
try {
  C
  // comandi che tentiamo di
  // e che possono lanciare
  // eccezioni
} catch (E1 exc) {
  // gestione dell'eccezione E1
} catch (E2 exc) {
  // gestione dell'eccezione E2
} ...
  ...
} finally {
  // codice eseguito sempre
  // a prescindere dal fatto
  // che C abbia o meno
  // lanciato eccezioni
}
```

Se in C (o in un metodo invocato da C) viene lanciata un'eccezione, l'esecuzione di C si interrompe e riprende in corrispondenza della **prima** clausola `catch` tale che il tipo (reale) dell'eccezione lanciata è sotto tipo di E_i

Se nessuna clausola `catch` è in grado di gestire l'eccezione, il metodo corrente viene terminato e l'eccezione viene **propagata** al metodo chiamante (dopo l'esecuzione della clausola `finally`, se presente)


TO
BE
CONTINUED!