

Eccezioni

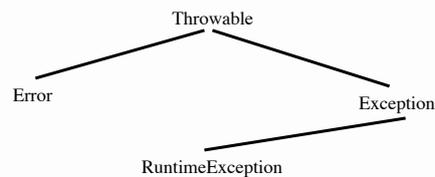
1

Le eccezioni in Java

- ✓ i tipi di eccezione sono particolari classi che
 - contengono solo il costruttore
 - ci possono essere più costruttori overloaded
 - sono definite in “moduli” separati da quelli che contengono i metodi che le possono sollevare
- ✓ le eccezioni sono oggetti
 - creati eseguendo new di un exception type
 - e quindi eseguendo il relativo costruttore
- ✓ esiste una gerarchia “predefinita” di tipi relativi alle eccezioni
 - nuovi tipi di eccezioni sono collocati nella gerarchia con l’usuale extends

2

La gerarchia di tipi per le eccezioni



- ✓ se un nuovo tipo di eccezione estende la classe `Exception`
 - l'eccezione è checked
- ✓ se un nuovo tipo di eccezione estende la classe `RuntimeException`
 - l'eccezione è unchecked

3

Eccezioni checked e unchecked

- ✓ se un metodo può sollevare una eccezione checked
 - deve elencarla nel suo header
 - che fa parte anche della specifica
 - altrimenti si verifica un errore a tempo di compilazione
- ✓ se un metodo può sollevare una eccezione unchecked
 - può non elencarla nel suo header
 - il suggerimento è di elencarla sempre, per rendere completa la specifica
- ✓ se un metodo chiamato da `p` ritorna sollevando una eccezione
 - se l'eccezione è checked
 - `p` deve gestire l'eccezione (try and catch, vedi dopo)
 - se l'eccezione (o uno dei suoi supertipi) è elencata tra quelle sollevabili da `p`, può essere propagata alla procedura che ha chiamato `p`
 - se l'eccezione è unchecked
 - può essere comunque gestita o propagata

4

Eccezioni primitive

- ✓ ne esistono numerose, sia checked che unchecked
 - `NullPointerException` e `IndexOutOfBoundsException` sono unchecked
 - `IOException` è checked

5

Definire tipi di eccezione

```
public class NuovoTipoDiEcc extends Exception {  
    public NuovoTipoDiEcc(string s) {super(s);}  
}
```

- ✓ è checked
- ✓ definisce solo un costruttore
 - come sempre invocato quando si crea una istanza con la `new`
 - il costruttore può avere parametri
- ✓ il corpo del costruttore riutilizza semplicemente il costruttore del supertipo
 - perché deve passargli il parametro
- ✓ una `new` di questa classe provoca la creazione di un nuovo oggetto che “contiene” la stringa passata come parametro

6

Costruire oggetti eccezione

```
public class NuovoTipoDiEcc extends Exception {  
    public NuovoTipoDiEcc(string s) {super(s);}  
}
```

- ✓ una new di questa classe provoca la creazione di un nuovo oggetto che “contiene” la stringa passata come parametro

```
Exception e = new NuovoTipoDiEcc ("Questa è la  
    ragione") ;  
String s = e.toString() ;
```

- ✓ la variabile s punta alla stringa
 “NuovoTipoDiEcc: Questa è la ragione”

7

Sollevare eccezioni

- ✓ una procedura può terminare
 - (ritorno normale) con un return
 - (ritorno di una eccezione) con un throw

```
public static int fact (int n) throws NonpositiveExc  
    // EFFECTS: se n>0, ritorna n!  
    // altrimenti solleva NonpositiveExc  
{ if (n <= 0) throw new NonPositiveExc("Num.fact");  
    ...}
```

- ✓ la stringa contenuta nell’eccezione è utile soprattutto quando il programma non è in grado di “gestire” l’eccezione
 - permette all’utente di identificare la procedura che la ha sollevata
 - può comparire nel messaggio di errore che si stampa subito prima di forzare la terminazione dell’esecuzione

8

Gestire eccezioni

- ✓ quando una procedura termina con un `throw`
 - l'esecuzione non riprende con quello che segue la chiamata
 - ma il controllo viene trasferito ad un pezzo di codice preposto alla gestione dell'eccezione
- ✓ due possibilità per la gestione
 - gestione esplicita quando l'eccezione è sollevata all'interno di uno statement `try`
 - in generale, quando si ritiene di poter recuperare uno stato consistente e di portare a termine una esecuzione quasi "normale"
 - gestione di default, mediante propagazione dell'eccezione alla procedura chiamante
 - possibile solo per eccezioni non checked o per eccezioni checked elencate nell'header della procedura che riceve l'eccezione

9

Gestione esplicita delle eccezioni

- ✓ gestione esplicita quando l'eccezione è sollevata all'interno di uno statement `try`
- ✓ codice per gestire l'eccezione `NonPositiveExc` eventualmente sollevata da una chiamata di `fact`

```
try { x = Num.fact (y); }  
catch (NonPositiveExc e) {  
    // qui possiamo usare e, cioè l'oggetto eccezione  
}
```
- ✓ la clausola `catch` non deve necessariamente identificare il tipo preciso dell'eccezione, ma basta un suo supertipo

```
try { x = Arrays.searchSorted (v, y); }  
catch (Exception e) { s.Println(e); return; }  
    // s è una PrintWriter
```
- ✓ segnala l'informazione sia su `NullPointerException` che su `NotFoundExc`

10

Try e Catch annidati

```
try { ...;
  try { x = Arrays.searchSorted (v, y); }
  catch (NullPointerException e) {
    throw new NotFoundExc ();}
  }
catch (NotFoundExc b ) {...}
```

- ✓ la clausola `catch` nel `try` più esterno cattura l'eccezione `NotFoundExc` se è sollevata da `searchSorted` o dalla clausola `catch` più interna

11

Catturare eccezioni unchecked

- ✓ le eccezioni unchecked sono difficili da catturare
 - una qualunque chiamata di procedura può sollevarle
 - difficile sapere da dove vengono

```
try { x = y[n]; i = Arrays.searchSorted (v, x); }
catch (IndexOutOfBoundsException e) {
  // cerchiamo di gestire l'eccezione pensando che sia
  // stata sollevata da x = y[n]
}
// continuiamo supponendo di aver risolto il problema
```

- ✓ ma l'eccezione poteva venire dalla chiamata a `searchSorted`
- ✓ l'unico modo per sapere con certezza da dove viene è restringere lo scope del comando `try`

12

Aspetti metodologici

- ✓ gestione delle eccezioni
 - riflessione
 - mascheramento
- ✓ quando usare le eccezioni
- ✓ come scegliere tra checked e unchecked
- ✓ defensive programming

13

Gestione delle eccezioni via riflessione

- ✓ se un metodo chiamato da p ritorna sollevando una eccezione, anche p termina sollevando un'eccezione
 - usando la propagazione automatica
 - della stessa eccezione (NullPointerException)
 - catturando l'eccezione e sollevandone un'altra
 - possibilmente diversa (EmptyException)

```
public static int min (int[] a) throws NullPointerException,
    EmptyException
    // EFFECTS: se a è null solleva NullPointerException
    // se a è vuoto solleva EmptyException
    // altrimenti ritorna il minimo valore in a
{int m;
try { m = a[0]}
catch(IndexOutOfBoundsException e) {
    throws new EmptyException("Arrays.min");}
for (int i = 1; i < a.length ; i++)
    if (a[i] < m) m = a[i];
return m;}
```

14

Gestione delle eccezioni via mascheramento

- ✓ se un metodo chiamato da p ritorna sollevando una eccezione, p gestisce l'eccezione e ritorna in modo normale

```
public static boolean sorted (int[] a) throws  
    NullPointerException  
    // EFFECTS: se a è null solleva NullPointerException  
    // se a è ordinato in senso crescente ritorna true  
    // altrimenti ritorna false  
{int prec;  
  try { prec = a[0]}  
  catch(IndexOutOfBoundsException e) { return true;}  
  for (int i = 1; i < a.length ; i++)  
    if (prec <= a[i]) prec = a[i]; else return false;  
  return true;}
```

- ✓ come nell'esempio precedente, usiamo le eccezioni (catturate) al posto di un test per verificare se a è vuoto

15

Quando usare le eccezioni

- ✓ le eccezioni non sono necessariamente errori
 - ma metodi per richiamare l'attenzione del chiamante su situazioni particolari (classificate dal progettista come eccezionali)
- ✓ comportamenti che sono errori ad un certo livello, possono non esserlo affatto a livelli di astrazione superiore
 - `IndexOutOfBoundsException` segnala chiaramente un errore all'interno dell'espressione `a[0]` ma non necessariamente per le procedure `min` e `sort`
- ✓ il compito primario delle eccezioni è di ridurre al minimo i vincoli della clausola `REQUIRES` nella specifica
 - dovrebbe restare solo se
 - la condizione è troppo complessa da verificare (efficienza)
 - il contesto d'uso limitato del metodo (private) ci permette di convincerci che tutte le chiamate della procedura la soddisfano
- ✓ vanno usate per evitare di codificare informazione su terminazioni particolari nel normale risultato

16

Checked o unchecked

- ✓ le eccezioni checked offrono una maggiore protezione dagli errori
 - sono più facili da catturare
 - il compilatore controlla che l'utente le gestisca esplicitamente o per lo meno le elenchi nell'header, prevedendone una possibile propagazione automatica
 - se non è così, viene segnalato un errore
- ✓ le eccezioni checked possono essere (per la stessa ragione) pesanti da gestire in quelle situazioni in cui siamo ragionevolmente sicuri che l'eccezione non verrà sollevata
 - perché esiste un modo conveniente ed efficiente di evitarla
 - per il contesto di uso limitato
 - solo in questi casi si dovrebbe optare per una eccezione unchecked

17

Defensive programming

- ✓ l'uso delle eccezioni facilita uno stile di progettazione e programmazione che protegge rispetto agli errori
 - anche se non sempre un'eccezione segnala un errore
- ✓ fornisce una metodologia che permette di riportare situazioni di errore in modo ordinato
 - senza disperdere tale compito nel codice che implementa l'algoritmo
- ✓ nella programmazione defensive, si incoraggia il programmatore a verificare l'assenza di errori ogniqualvolta ciò sia possibile
 - ed a riportarli usando il meccanismo delle eccezioni
 - un caso importante legato alle procedure parziali

18

Quando un metodo non soddisfa la sua preconditione

- ✓ con le eccezioni i metodi tendono a diventare totali
 - ma non è sempre possibile
- ✓ Chi invoca il metodo dovrebbe farsi carico di effettuare tale controllo
 - sollevando una eccezione
 - questa eccezione può essere catturata, magari ad un livello superiore
 - si suggerisce di usare in questi casi una eccezione generica unchecked `FileNotFoundException`

19

```
public void storeDataFromUrl(String url){
    try {
        String data = readDataFromUrl(url);
    } catch (BadUrlException e) {
        e.printStackTrace();
    }
}

public String readDataFromUrl(String url)
throws BadUrlException{
    if(isUrlBad(url)){
        throw new BadUrlException("Bad URL: " + url);
    }

    String data = null;
    //read lots of data over HTTP and return
    //it as a String instance.

    return data;
}
```

20