Programmazione II

4. Eccezioni (PDJ 4)

Dragan Ahmetovic

Limiti delle procedure parziali

Caratteristiche di una procedura parziale

Una **procedura parziale** non descrive il comportamento per input non gestiti.

Il comportamento **non è definito** quindi possono avvenire cose diverse, ad esempio:

- La procedura e il programma terminano in maniera improvvisa
- La procedura cicla senza terminare mai
- La procedura restituisce un **valore errato**
  - Particolarmente problematico perché l'**errore potrebbe non essere notato**
  - Potrebbe **propagarsi causando danni irreversibili**

Il programma **non è robusto**

Limiti delle procedure parziali

Graceful Degradation

Proprietà desiderabile di un programma. Implica che:

- Il programma dovrebbe essere in grado di gestire l'errore
  - o recuperando il comportamento desiderato
  - o terminando **senza causare danni** e informando l'utente
- Mantenere un comportamento **ben definito**

Rendere le procedure totali

- Incrementa la **robustezza** del programma gestendo i casi d'errore
- Non sempre possibile o auspicabile
  - Se incide troppo sulla performance
  - Se richiede troppo sforzo implementativo
  - Se non serve ai fini dello sviluppo

Meccanismi di notifica dell'errore

Valori return speciali

Posso gestire i casi d'errore sui valori in input restituendo output particolari.

Esempio:

```
1 public static double square(double n){
2 //EFFECTS: if n <= 0 returns 0,
3 //          otherwise returns square root of n
```

Problemi:

- Non è possibile se tutto il codominio è utilizzato
- Alto rischio di **runtime errors**
- Alto rischio di **propagazione**
- Alto costo di **gestione dell'errore**

Meccanismi di notifica dell'errore

Esempio:

```
1  public static int searchSorted (int[] a, int x) {
2  // REQUIRES: a is sorted in ascending order
3  // EFFECTS: If x is in a, returns an index where x is stored; otherwise, returns -1.
4
5  . . .
6
7  int startIndex = searchSorted(a, x) // searching x in a but there is none
8
9  //I want to print everything after x, but if there is none the program will crash
10 for(i = startIndex; i < a.length; a++)
11   System.out.println(a[i]);
12
13 -----------------
14
15 //need to check
16 if(startIndex) {
17   for(i = startIndex; i < a.length; a++)
18     System.out.println(a[i]);
```

Meccanismi di notifica dell'errore

Esempio:

```
1  public static int factorial (intn) {
2  // EFFECTS: If n>0 returns n!;
3  //          else returns 0
4
5  . . .
6
7  sumOfFactorials += factorial(y) //if y is negative this will work but calculation is wrong
8
9  ---------------------------
10
11 //I can solve this as follows but it becomes long and confusing
12 int tmpFac = factorial(y);
13
14 if(tmpFac > 0)
15   sumOfFactorials += tmpFac;
```

Meccanismi di notifica dell'errore

Comma OK

In alcuni linguaggi è possibile restituire più valori (es: GO)

Uno dei valori può essere utilizzato per segnalare comportamenti anomali o errori

In java è possibile fare **wrapping** di più valori di return in un unico tipo di ritorno

Problemi:

- Bisogna ricordarsi di **verificare** se c'è stato un errore
- Altissimo rischio di **propagazione**
- Alto rischio di **runtime errors**

Meccanismi di notifica dell'errore

Esempio:

```
if temp["room 3"] <= 0 { //Is temperature 0 or there was no reading avaliable?
  turn_heater_on()
}

------------------------------

temperature, ok = temp["room 3"]

if ok && temperature <= 0 { //I have to remember to check ok value
  turn_heater_on()
}
```

Eccezioni

Che cosa sono?

Meccanismo (parzialmente) separato dal control flow del programma per la gestione degli **eventi eccezionali**

- Un evento eccezionale causa un'interruzione della procedura
- È **indipendente** dal codominio della procedura (dal valore di return)
- Costringe a **gestire** l'iregolarità:
  - **Non si propaga** perchè è sicuramente individuato.
- È possibile gestire l'irregolarità!

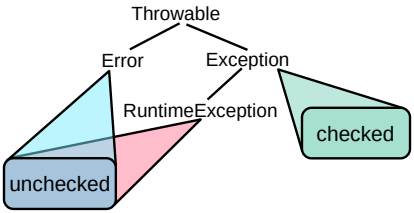
Eccezioni

Tipi di eccezioni

Sono comunque tipi oggetto (sottotipo **Throwable**). possiamo crearne di nuove!

Gerarchia con due gruppi principali:

- **Error** - errori fatali nell'esecuzione del programma
- **Exception** - comportamenti inattesi ma gestibili nell'esecuzione del programma
  - **RuntimeException** - avvenimenti imprevedibili
  - **Altre Exception** - avvenimenti prevedibili



Eccezioni

Unchecked exceptions

- Error e RuntimeException
- Eventi fuori dal controllo del programmatore
- Irrealistico chiedere al programmatore di gestirle tutte
  - Ma Liskov chiede di farlo il più possibile per garantire robustezza e specifiche affidabili

Checked exceptions

- Tutte le altre
- Mi aspetto che il programmatore le gestisca nel codice
- Altrimenti **compile error**

Eccezioni

Come si usano?

Trattamento esplicito:

- Rinchiudere le espressioni che possono causare eccezioni in un blocco `try {...}`
- Seguito da un blocco `catch(<TipoEccezione> e) {...}` che uso per gestire l'eccezione
  - All'interno del blocco posso **cercare di sistemare** l'errore
- È permesso un blocco finale `finally {...}`. viene eseguito sempre, anche se try va bene
  - Serve per chiudere le risorse aperte
  - Alternativa: `try(<inizializzazione risorsa 1>; <inizializzazione risorsa 2>; ...) {}`

Eccezioni

Esempio:

```
1  Scanner scanner = null;
2
3  try {
4    scanner = new Scanner(new File("test.txt"));
5    while (scanner.hasNext())
6      System.out.println(scanner.nextLine());
7  } catch (FileNotFoundException e) {
8    e.printStackTrace();
9  } finally {
10   if (scanner != null)
11     scanner.close();
12 }
```

Eccezioni

Esempio:

```
1 try (Scanner scanner = new Scanner(new File("test.txt"))) {
2   while (scanner.hasNext())
3     System.out.println(scanner.nextLine());
4 } catch (FileNotFoundException e) {
5   e.printStackTrace();
6 }
```

Eccezioni

Come si specifica?

Sono necessarie modifiche alla **signature** e alla **specifica** dei metodi

- keyword **"throws"** nella signature, seguita dai tipi di eccezione che potrebbero accadere
  - Liskov suggerisce di riportare anche le eccezioni unchecked per rendere chiaro il comportamento
- Deve esserci la descrizione di ciascuno dei casi in **EFFECTS**
- Se l'eccezione avviene per alcuni valori del dominio, questi contano come validi.
  - Non bisogna specificarli come vincoli in **REQUIRES**
- se vi sono modifiche agli input o effetti collaterali, nella clausola **MODIFIES** bisogna rendere chiaro se e in quali eccezioni avvengono le modifiche.

Eccezioni

Esempio:

```
1  public static int fact(int n) throws NonPositiveException
2  // EFFECTS: If n is non-positive, throws NonPositiveException;
3  //          else returns the factorial of n.
4
5  public static int search(int[] a, int x) throws NullPointerException, NotFoundException
6  // REQUIRES: a is sorted
7  // EFFECTS: If a is null throws NullPointerException;
8  //          else if x is not in a, throws NotFoundException;
9  //          else returns i such that a[i] = x.
10
11 public static void addMax(List<Integer> t, int x) throws NullPointerException,
       TooBigException
12 // REQUIRES: All elements of v are Integers.
13 // MODIFIES: v
14 // EFFECTS: If v is null throws NullPointerException;
15 //          if v contains an element larger than x throws TooBigException;
16 //          else adds x to v.
```

Eccezioni

Come si lanciano?

Si usa l'istruzione **throw**:

- "throw new Exception();"
- "throw new Exception("Messaggio d'errore");"

Utile lanciare eccezioni di **tipo appropriato**:

- Tra quelle esistenti (NullPointerException, IllegalArgumentException...)
- Fornire un **messaggio d'errore** utile per gestirlo (su crash o da log)
  - Info su classe, metodo, parametri e contesto d'uso
- Creare **eccezioni nuove** più appropriate alla situazione

Eccezioni

Esempio:

```
1 public static int findMin(int[] a) throws NullPointerException, IllegalArgumentException {
2 // EFFECTS: Returns the smallest value in a
3 //          if a is null throws NullPointerException
4 //          else if a is empty, throws IllegalArgumentException
5   if(arr == null)
6     throw new NullPointerException("Can't handle null arrays");
7   if(arr.length == 0)
8     throw new IllegalArgumentException("Can't handle zero-length arrays.");
9 . . .
```

Eccezioni

Come si definiscono?

Sono oggetti qualsiasi; serve estendere una classe base

- `public class NewKindOfException extends Exception` (checked)
- `public class NewKindOfException extends RuntimeException` (unchecked)

Il funzionamento è gestito dalla rispettiva **superclasse**

Bisogna solamente aggiungere i **costruttori**:

- "public NewKindOfException() {super();}"
- "public NewKindOfException(String s) {super(s);}"

Eccezioni

Esempio:

```
1  public class NewCheckedException extends Exception {
2
3    public NewCheckedException() {
4      super(); //this calls the constructor of the superclass
5    }
6
7    public NewCheckedException(String s) {
8      super(s); //this calls the constructor of the superclass with s
9    }
10 }
11
12 public class NewUncheckedException extends RuntimeException {
13
14   public NewUncheckedException() {
15     super(); //this calls the constructor of the superclass
16   }
17
18   public NewUncheckedException(String s) {
19     super(s); //this calls the constructor of the superclass with s
20   }
21 }
```

Eccezioni

Checked o Unchecked?

La scelta dipende da **aspettative sull'uso** dell'eccezione:

- Se è realistico che **l'eccezione accada** allora è utile averla **checked**
- Se è **facile evitare** che accada oppure il contesto d'uso è controllato ok se è **unchecked**

Reflecting e Masking

Quando passare l'eccezione al livello superiore e quando trattarla localmente?

- Se posso gestirla **localmente** lo faccio e riprendo l'esecuzione normale (**Masking**)
- Se **non ho contesto** per trattare l'eccezione localmente la lascio passare (**Reflecting**)
- Posso anche **catturarla** e **lanciare una nuova eccezione** per fornire più contesto all'utente.

Eccezioni

Esempio:

```
1  public static int min (int[] a) throws NullPointerException, EmptyException {
2  // EFFECTS: If a is null throws NullPointerException;
3  //          else if a is empty throws EmptyException;
4  //          else returns the minimum value of a
5    if (a.length == 0)
6      throw new EmptyException("Arrays.min");
7
8    int m = a[0];
9
10   for(int i = 1; i < a.length; i++)
11     if (a[i] < m)
12       m = a[i];
13
14   return m;
15 }
```

Eccezioni

Esempio:

```
1  public static void readFile(String filename) {
2  // MODIFIES: filename file, System.out
3  // EFFECTS: if the file exists open it for reading and print its content to System.out;
4  //          otherwise print that the file does not exist
5    Scanner scanner = null;
6
7    try {
8      scanner = new Scanner(new File(filename));
9    } catch (FileNotFoundException e) {
10     System.out.println("file " + filename + "does not exist")
11   } finally {
12     if (scanner != null) {
13       scanner.close();
14     }
15   }
16
17   while (scanner.hasNext()) {
18     System.out.println(scanner.nextLine());
19   }
20 }
```

Programmazione II

4. Eccezioni (PDJ 4)

Dragan Ahmetovic