Java: Eccezioni

Damiano Macedonio

Dipartimento di Informatica, Università degli Studi di Verona

Corso di Programmazione per Bioformatica

lezione del 30 maggio 2014

Un programma diviso in sezioni distinte

- Un approccio alla scrittura di un programma consiste nel presupporre che durante la sua esecuzione non si presentino situazioni anomale (o eccezionali).
- Una volta che il programma funziona per il caso normale, si aggiunge il codice che gestisce i casi eccezionali.
- Si suddivide così il programma in due sotto-attività più piccole e quindi più semplici da trattare.
- Java fornisce gli strumenti necessari per attuare questo approccio.

Nomenclatura

Exception

Un'eccezione è un *oggetto* che segnala l'accadere di un evento anomalo (o eccezionale) durante l'esecuzione di un programma.

Throwing an Exception

Il processo di creazione di un oggetto eccezione è chiamato lancio di un'eccezione.

Handle the Exception

In un'altra parte del programma si può inserire il codice che si occupa dell'evento eccezionale, ovvero che gestisce l'eccezione.

Un esempio: calcolo dei consumi

```
1  public class Consumi {
2
3      public static void main(String[] args) {
4          java.util.Scanner tastiera = new java.util.Scanner(System.in);
5          int kilometri, litri, distanza;
6
7          System.out.print("Inserire i kilometri percorsi: ");
8          kilometri = tastiera.nextInt();
9
10         System.out.print("Inserire i litri di benzina consumati: ");
11         litri = tastiera.nextInt();
12
13         distanza = kilometri / litri;
14
15         System.out.println("La tua auto fa " + distanza + " kilometri al litro");
16
17         System.out.println("... fine del programma.");
18     }
19 }
```

Se l'utente immette 0 litri...

Il programma potrebbe fare una divisione per zero!

Esempio di esecuzione del programma:

```
Inserire i kilometri percorsi: 8
Inserire i litri di benzina consumati: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Consumi.main(Consumi.java:13)
```

Per evitare il problema abbiamo a disposizione due approcci:

1. usare un blocco `if-else`
2. usare un blocco `try-catch`

(1) Controlliamo il valore immesso

Quando l'utente immette il valore per `litri`, il programma si accerta che tale valore non sia nullo (o negativo). In tal caso, non viene eseguita la divisione.

```
litri = tastiera.nextInt();

if (litri <= 0) {
    ...
}
else {
    distanza = kilometri / litri;
    ...
}
```

(1) Controlliamo il valore immesso

```
1  public class Consumi {
2
3      public static void main(String[] args) {
4          java.util.Scanner tastiera = new java.util.Scanner(System.in);
5          int kilometri, litri, distanza;
6
7          System.out.print("Inserire i kilometri percorsi: ");
8          kilometri = tastiera.nextInt();
9
10         System.out.print("Inserire i litri di benzina consumati: ");
11         litri = tastiera.nextInt();
12
13         if (litri <= 0) {
14             System.out.println("Impossibile: la tua auto non consuma benzina?!?");
15             System.out.println("Controlla meglio...");
16         } else {
17             distanza = kilometri / litri;
18             System.out.println("La tua auto fa " + distanza + " kilometri al
                   litro");
19         }
20
21         System.out.println("... fine del programma.");
22     }
23 }
```

In questo modo...

La divisione per zero non viene mai effettuata!

Esempio di esecuzione del programma:

```
Inserire i kilometri percorsi: 8
Inserire i litri di benzina consumati: 0
Impossibile: la tua auto non consuma benzina?!?
Controlla meglio...
... fine del programma.
```

(2) Dividiamo comunque nel caso, gestiamo l'eccezione

Una divisione per zero produce un'eccezione. Invece che evitare l'esecuzione della divisione, la si può effettuare e si reagisce opportunamente all'eventuale eccezione.

```
1   try {
2       ...
3
4       distanza = kilometri / litri;
5
6       System.out.println("La tua auto fa " + distanza + " kilometri al litro");
7
8   } catch(Exception e) {
9       System.out.println("c'e' stato un problema: " + e.getMessage());
10      System.out.println("Possibile che la tua auto non consumi?!?");
11  }
```

(2) Dividiamo comunque...

```
1  public class Consumi {
2
3    public static void main(String[] args) {
4      java.util.Scanner tastiera = new java.util.Scanner(System.in);
5      int kilometri, litri, distanza;
6
7      try {
8        System.out.print("Inserire i kilometri percorsi: ");
9        kilometri = tastiera.nextInt();
10       System.out.print("Inserire i litri di benzina consumati: ");
11       litri = tastiera.nextInt();
12
13       distanza = kilometri / litri;
14
15       System.out.println("La tua auto fa " + distanza + " kilometri al litro");
16
17     } catch (Exception e) {
18       System.out.println("Forse hai un problema: " + e.getMessage());
19       System.out.println("Possibile che la tua auto non consumi?!?");
20     }
21
22     System.out.println("... fine del programma.");
23   }
24 }
```

In questo modo...

La divisione per zero può succedere, ma viene gestita!

Esempi di esecuzione del programma:

```
Inserire i kilometri percorsi: 8
Inserire i litri di benzina consumati: 2
La tua auto fa 4 kilometri al litro
... fine del programma.

Inserire i kilometri percorsi: 8
Inserire i litri di benzina consumati: 0
Forse hai un problema: / by zero
Possibile che la tua auto non consumi?!?
... fine del programma.
```

Blocco `try-catch`

- Le situazioni normali vengono gestite dal codice nel blocco `try`.
- Il codice nel blocco `catch` viene utilizzato solamente quando viene lanciata un'eccezione.
- Un'eccezione è un oggetto che viene creato con un messaggio.
- Il messaggio può essere recuperato con il metodo `getMessage()`.

Flusso del programma - con eccezione

- Un blocco `try` contiene un frammento di codice che può lanciare un'eccezione.
- Quando viene lanciata un'eccezione, l'esecuzione del blocco `try` termina: non viene eseguita nessuna delle istruzioni del blocco `try` poste dopo l'istruzione che ha lanciato l'eccezione.
- Viene eseguita la porzione di codice del blocco `catch` appropriato.
- Al termine dell'esecuzione del codice contenuto nel blocco `catch`, il programma prosegue con il codice posto all'esterno dell'ultimo blocco `catch`.

Un blocco `catch` è legato al solo blocco `try` immediatamente precedente. Se un'eccezione viene lanciata e catturata, l'oggetto eccezione viene assegnato all'identificativo del parametro del blocco `catch`.

Flusso del programma - senza eccezione

- Quando un blocco `try` viene eseguito normalmente fino al completamento, senza generare alcuna eccezione, l'esecuzione del programma prosegue con il codice che si trova dopo l'ultimo blocco `catch`.
- Se non viene generata alcuna eccezione, tutti i relativi blocchi `catch` vengono ignorati.

Lanciare un'eccezione (istruzione `throw`)

```
try {
    ...
    if (litri <= 0)
        throw new Exception("hai tentato di dividere per zero!");
    else
        distanza = kilometri / litri;
    ...
} catch (Exception e) {
    System.out.println("Forse hai un problema: " + e.getMessage());
    ...
}
```

- Se viene eseguita, l'istruzione `trow` crea un nuovo oggetto della classe predefinita `Exception` e lancia (`throws`) l'oggetto creato.
- La stringa `hai tentato di dividere per zero!` costituisce l'argomento del costruttore della classe `Exception`.
- L'eccezione così creata memorizza tale stringa in una sua variabile di istanza, in modo che possa essere successivamente recuperata col metodo `getMessage()`.

Lanciare un'eccezione (istruzione `throw`)

```
1  public class Consumi {
2
3    public static void main(String[] args) {
4      java.util.Scanner tastiera = new java.util.Scanner(System.in);
5      int kilometri, litri, distanza;
6
7      try {
8        System.out.print("Inserire i kilometri percorsi: ");
9        kilometri = tastiera.nextInt();
10       System.out.print("Inserire i litri di benzina consumati: ");
11       litri = tastiera.nextInt();
12
13       if (litri <= 0)
14         throw new Exception("hai tentato di dividere per zero!");
15
16       distanza = kilometri / litri;
17       System.out.println("La tua auto fa " + distanza + " kilometri al litro");
18     } catch (Exception e) {
19       System.out.println("Forse hai un problema: " + e.getMessage());
20       System.out.println("Possibile che la tua auto non consumi?!?");
21     }
22     System.out.println("... fine del programma.");
23   }
24 }
```

Esempi di esecuzione

```
Inserire i kilometri percorsi: 20
Inserire i litri di benzina consumati: 3
La tua auto fa 6 kilometri al litro
... fine del programma.

Inserire i kilometri percorsi: 20
Inserire i litri di benzina consumati: 0
Forse hai un problema: hai tentato di dividere per zero!
Possibile che la tua auto non consumi?!?
... fine del programma.
```

Classi di eccezioni predefinite

I metodi delle classi predefinite possono lanciare eccezioni che appartengono a classi predefinite all'interno della Java Class Library. Se si utilizza uno di questi metodi, è possibile inserire la sua invocazione in un blocco `try` e utilizzare il blocco `catch` per catturare la sua eventuale eccezione.

`ArithmeticException` Operazione aritmetica non consentita.

`InputMismatchException` Valore immesso errato

`BadStringOperationException` Operazione non consentita.

`ClassNotFoundException` Classe non trovata.

`IOException` Errore in input o output.

`NoSuchMethodException` Metodo inesistente.

Sono tutte sottoclassi di `Exception`.

Meglio catturare specifiche eccezioni

Sebbene sia possibile utilizzare la classe `Exception` in un blocco `catch` è più utile catturare eccezioni più specifiche.

Un'altra possibile esecuzione del programma

```
Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3.0
Forse hai un problema: null
Possibile che la tua auto non consumi?!?
... fine del programma.
```

L'eccezione lanciata riguarda l'input immesso dall'utente. Il metodo `nextInt()` della classe `Scanner` si aspetta la rappresentazione di un intero. Non trovandola lancia una `InputMismatchException`. L'eccezione viene catturata dal blocco `catch (Exception e) {...}`

Catturiamo l'eccezione `ArithmeticException`

```
1  public class Consumi {
2
3    public static void main(String[] args) {
4      java.util.Scanner tastiera = new java.util.Scanner(System.in);
5      int kilometri, litri, distanza;
6
7      try {
8        System.out.print("Inserire i kilometri percorsi: ");
9        kilometri = tastiera.nextInt();
10       System.out.print("Inserire i litri di benzina consumati: ");
11       litri = tastiera.nextInt();
12
13       distanza = kilometri / litri;
14
15       System.out.println("La tua auto fa " + distanza + " kilometri al litro");
16     } catch (ArithmeticException e) { // selezioniamo l'eccezione
17       System.out.println("Forse hai un problema: " + e.getMessage());
18       System.out.println("Possibile che la tua auto non consumi?!?");
19     }
20     System.out.println("... fine del programma.");
21   }
22 }
```

In questo modo...

Viene catturata solo l'eccezione specifica, le altre vengono rilanciate.

```
Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3
La tua auto fa 10 kilometri al litro
... fine del programma.

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 0
Forse hai un problema: / by zero
Possibile che la tua auto non consumi?!?
... fine del programma.

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3.0
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:909)
    at java.util.Scanner.next(Scanner.java:1530)
    at java.util.Scanner.nextInt(Scanner.java:2160)
    at java.util.Scanner.nextInt(Scanner.java:2119)
    at Consumi.main(Consumi.java:11)
```

Blocchi `catch` multipli

- Un blocco `try` può potenzialmente lanciare un numero qualsiasi di eccezioni, che possono essere di differenti tipi.
- Ogni blocco `catch` può catturare eccezioni di un solo tipo.
- È possibile catturare più tipi di eccezioni inserendo più blocchi `catch` dopo un blocco `try`.

Blocchi `catch` multipli

```
1  import java.util.InputMismatchException; // import!
2
3  public class Consumi {
4
5    public static void main(String[] args) {
6      java.util.Scanner tastiera = new java.util.Scanner(System.in);
7      int kilometri, litri, distanza;
8
9      try {
10       System.out.print("Inserire i kilometri percorsi: ");
11       kilometri = tastiera.nextInt();
12       System.out.print("Inserire i litri di benzina consumati: ");
13       litri = tastiera.nextInt();
14       distanza = kilometri / litri;
15       System.out.println("La tua auto fa " + distanza + " kilometri al litro");
16
17     } catch (ArithmeticException e) {
18       System.out.println("Forse hai un problema: " + e.getMessage());
19       System.out.println("Possibile che la tua auto non consumi?!?");
20     } catch (InputMismatchException e) {
21       System.out.println("Hai immesso un valore inatteso, riprova!");
22     }
23     System.out.println("... fine del programma.");
24   }
25 }
```

Esempi di esecuzione

```
Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3
La tua auto fa 10 kilometri al litro
... fine del programma.

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 0
Forse hai un problema: / by zero
Possibile che la tua auto non consumi?!?
... fine del programma.

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3.0
Hai immesso un valore inatteso, riprova!
... fine del programma.
```

Definire nuove classi di eccezioni

È possibile definire nuove classi di eccezioni come sottoclassi di una classe di eccezione già definita.

Linee guida

- Se non ci sono particolari esigenze, utilizzate `Exception` come classe base.
- Definite almeno due costruttori: uno di default ed uno con un solo parametro di tipo `String` (il messaggio).
- Iniziate ogni definizione di costruttore con una chiamata a `super()` per inizializzare il messaggio.
- Il metodo `getMessage()` ereditato non dovrebbe essere ridefinito.
- Normalmente non è necessario definire nessun altro metodo, anche se sarebbe lecito farlo.

Due esempi

```java
// DivisionePerZeroException.java
public class DivisionePerZeroException extends Exception {

    public DivisionePerZeroException() {
        super("divisione per zero!");
    }

    public DivisionePerZeroException(String messaggio) {
        super(messaggio);
    }
}
```

```java
// DivisionePerValoreNegativoException.java
public class DivisionePerValoreNegativoException extends Exception {

    public DivisionePerValoreNegativoException() {
        super("divisione per valore negativo!");
    }

    DivisionePerValoreNegativoException(String messaggio) {
        super(messaggio);
    }
}
```

Metodi che lanciano eccezioni (clausola `throws`)

```
public static int dividi(int n, int d) throws DivisionePerZeroException {
  if (denominatore > 0)
    return n / d;
  else
    throw new DivisionePerZeroException();
}
```

- Il metodo `dividi()` lancia un'eccezione che però non cattura.
- La gestione dell'eccezione dipende da chi utilizza il metodo.
- L'invocazione del metodo deve essere inserita in un blocco `try`, seguito da un blocco `catch` che catturi l'eccezione.
- La clausola `throws` dichiara che un'invocazione del metodo `dividi()` può lanciare una `Exception`.

Lanciare eccezioni multiple

Una clausola `throws` può contenere più tipi di eccezioni separati da una virgola

```
public static int dividi(int n, int d) throws DivisionePerZeroException,
     DivisionePerValoreNegativoException {
  if (d > 0)
    return n / d;
  else if (d == 0)
    throw new DivisionePerZeroException();
  else // d < 0
    throw new DivisionePerValoreNegativoException();
}
```

Invocazione di `dividi()`

```
public static void main(String[] args) {
  java.util.Scanner tastiera = new java.util.Scanner(System.in);
  int kilometri, litri, distanza;

  try {
    System.out.print("Inserire i kilometri percorsi: ");
    kilometri = tastiera.nextInt();
    System.out.print("Inserire i litri di benzina consumati: ");
    litri = tastiera.nextInt();
    distanza = dividi(kilometri, litri);
    System.out.println("La tua auto fa " + distanza + " kilometri al litro");

  } catch (InputMismatchException e) {
    System.out.println("Hai immesso un valore inatteso, riprova!");

  } catch (DivisionePerZeroException e) {
    System.out.println("Forse hai un problema: " + e.getMessage());
    System.out.println("Possibile che la tua auto non consumi?!?");

  } catch (DivisionePerValoreNegativoException e) {
    System.out.println("Forse hai un problema: " + e.getMessage());
    System.out.println("La tua auto produce benzina?!?");
  }
  System.out.println("... fine del programma.");
}
```

Possibili esecuzioni del programma

```
Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3
La tua auto fa 10 kilometri al litro
... fine del programma

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 0
Forse hai un problema: divisione per zero!
Possibile che la tua auto non consumi?!?
... fine del programma.

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: -3
Forse hai un problema: divisione per valore negativo!
La tua auto produce benzina?!?
... fine del programma.

Inserire i kilometri percorsi: 30
Inserire i litri di benzina consumati: 3.0
Hai immesso un valore inatteso, riprova!
... fine del programma.
```

Catturare prima l'eccezione più specifica!

Come verrebbero catturate le eccezioni in questo caso?

```
  try {
    ...
  } catch (Exception e) {
    System.out.println("Ho catturato una eccezione: " + e.getMessage());
  } catch (InputMismatchException e) {
    System.out.println("Hai immesso un valore inatteso, riprova!");
  } catch (DivisionePerZeroException e) {
    System.out.println("Forse hai un problema: " + e.getMessage());
    System.out.println("Possibile che la tua auto non consumi?!?");
  } catch (DivisionePerValoreNegativoException e) {
    System.out.println("Forse hai un problema: " + e.getMessage());
    System.out.println("La tua auto produce benzina?!?");
  }
  System.out.println("... fine del programma.");
}
```

Il primo blocco `catch` catturerebbe ogni tipo di eccezione, tutti gli altri blocchi sarebbero inutili: il loro codice non verrebbe mai raggiunto dal programma (unreachable catch block).

Una eccezione può venire rilanciata

Esempio

```
public void methodA() throws Exception {
    ...
    throw new Exception();
    ...
}

public void methodB() throws Exception {
    ...
    methodA();
    ...
}
```

`methodB()` rilancia l'eventuale eccezione lanciata da `methodA()` e trasferisce la gestione dell'eccezione al qualsiasi metodo chiami `methodB()`.

In un programma ben scritto, ogni eccezione sollevata dovrebbe prima o poi essere catturata da un blocco `catch`.

Blocco `finally`

È possibile inserire un blocco `finally` dopo una sequenza di blocchi `catch`. Il codice del blocco `finally` viene eseguito indipendentemente dal fatto che l'eccezione venga lanciata.

Blocco `try-catch-finally`: flusso del programma

- Se il blocco `try` viene eseguito completamente senza eccezioni, il blocco `finally` viene eseguito dopo il `try`
- Se viene lanciata un'eccezione dal blocco `try` e viene catturata da uno dei blocchi `catch` seguenti, il blocco `finally` viene eseguito al termine del `catch`
- Se viene lanciata un'eccezione dal blocco `try` ma non esiste un blocco `catch` che possa catturarla, viene immediatamente eseguito il blocco `finally` e quindi il metodo termina rilanciando l'eccezione al metodo chiamante.

Esempio

```
1  public static void main(String[]args) throws DivisionePerValoreNegativoException {
2      java.util.Scanner tastiera = new java.util.Scanner(System.in);
3      System.out.print("immettere un valore: ");
4
5      int val = tastiera.nextInt();
6      try {
7          if (val > 0)
8              System.out.println("Eseguo la divisione");
9          else if (val == 0) {
10             System.out.println("lancio DivisionePerZeroException");
11             throw new DivisionePerZeroException();
12         } else {
13             System.out.println("lancio DivisionePerValoreNegativoException");
14             throw new DivisionePerValoreNegativoException();
15         }
16
17     } catch (DivisionePerZeroException e) {
18         System.out.println("Eseguo il blocco catch");
19
20     } finally {
21         System.out.println("Eseguo il blocco finally");
22     }
23     System.out.println("Eseguo il codice finale");
24 }
```

Possibili esecuzioni del programma

```
immettere un valore: 4
Eseguo la divisione
Eseguo il blocco finally
Eseguo il codice finale

immettere un valore: 0
lancio DivisionePerZeroException
Eseguo il blocco catch
Eseguo il blocco finally
Eseguo il codice finale

immettere un valore: -4
lancio DivisionePerValoreNegativoException
Eseguo il blocco finally
Exception in thread "main" DivisionePerValoreNegativoException: divisione per
     valore negativo! at Test.main(Test.java:18)
```

N.B.

Se viene lanciata un'eccezione ma non c'è un blocco `catch` che la catturi, il codice finale non viene eseguito.

Tipi di eccezioni

Eccezioni controllate

Devono essere catturate da un blocco `catch` oppure dichiarate da una clausola `throws`.

Eccezioni non controllate

Possono non essere catturata da un blocco `catch` oppure dichiarate da una clausola `throws`.

Eccezioni controllate (checked exceptions)

- Devono essere catturate da un blocco `catch` oppure dichiarate da una clausola `throws`.
- Indicano la presenza di seri problemi che potrebbero causare la terminazione del programma.
- Alcuni esempi:

  `BadStringOperationException`
  `ClassNotFoundException`
  `IOException`
  `NoSuchMethodException`

Eccezioni non controllate (unchecked exceptions)

- Dette anche eccezioni run-time.
- Possono non essere catturate da un blocco `catch` oppure dichiarate da una clausola `throws`.
- Indicano che nel codice c'è qualcosa di sbagliato, che dovrebbe essere corretto.
- Normalmente per queste eccezioni non si è scritta un'istruzione `throw`.
- Solitamente generate dalla valutazione di un'espressione o lanciate da un metodo di una classe predefinita.
- Per queste eccezioni è necessario correggere il codice e non aggiungere un blocco `catch`.
- Un'eccezione run-time non catturata causa la terminazione del programma.
- Alcuni esempi: `ArithmeticException`, `InputMismatchException`, `ArrayIndexOutOfBoundException`.

Errori

- Un errore è un oggetto della classe `Error`.
- Non è un eccezione, ma si comporta in maniera simile.
- Gli oggetti della classe `Error` sono simili a *eccezioni non controllate*: non è necessario catturarli o dichiararli in una clausola `throws`, anche se questo è comunque possibile.
- Gli errori sono il più delle volte fuori dal controllo del programmatore.

Esempio

Si verifica un `OutOfMemoryError` quando il programma ha esaurito la memoria disponibile. Questo significa che si deve modificare il programma affinché utilizzi meno memoria o cambiare le impostazioni affinché Java utilizzi più memoria. L'aggiunta di un blocco `catch` non sarà di alcun aiuto.

Gerarchia delle eccezioni