Eccezioni

Situazioni anomale a run-time

- Java prevede un sofisticato utilizzo dei tipi (primitivi e classi) che consente di individuare molti errori al momento della compilazione del programma (prima dell'esecuzione vera e propria)
- Ciò nonostante si possono verificare varie situazioni impreviste o anomale durante l'esecuzione del programma che possono causare l'interruzione del programma stesso
- Ad esempio:
    - Tentativi di accedere a posizioni di un array che sono fuori dai limiti
    - Errori aritmetici (divisione per zero, ...)
    - Errori di formato: (errore di input dell'utente)

Le eccezioni si dividono in

Checked (o controllate) per le quali il compilatore richiede che ci sia un gestore

- **Controllate**
    - Istanze di RuntimeException o delle sue sottoclassi
    - Il compilatore si assicura esplicitamente che quando un metodo solleva una'eccezione la tratti esplicitamente
    - Questo può essere fatto mediante i costrutti try-catch o throws
    - In caso contrario segnala un errore di compilazione
    - Le eccezioni controllate vincolano il programmatore ad occuparsi della loro gestione
    - Le eccezioni controllate possono rendere troppo pesante la scrittura del codice

Unchecked (o non controllate) per le quali il gestore non è obbligatorio

- Per essere unchecked un'eccezione *deve essere una sottoclasse di* ***RuntimeException***, altrimenti è checked
- **Non controllate**
    - Sono tutte le altre eccezioni, ovvero istanze di Exception ma non di RuntimeException
    - L'eccezione può non essere gestita esplicitamente dal codice
    - Viene "passata" automaticamente da metodo chiamato a metodo chiamante

---

Esempi tipici di eccezioni checked:

- le eccezioni che descrivono errori di input/output
    - lettura o scrittura su file,
    - comunicazione via rete, ecc...
- le eccezioni definite dal programmatore

La gerarchia delle eccezioni

La classe **Exception** descrive un'eccezione generica, situazioni anomale più specifiche sono descritte dalle sottoclassi di Exception

Le RuntimeException comprese nel pacchetto java.lang

Eccezione	Significato
ArithmeticException	Operazione matematica non valida.
ArrayIndexOutOfBoundsException	L'indice usato in un array non è valido.
ArrayStoreException	Incompatibilità di tipo durante la assegnazione di un elemento di un array.
ClassCastException	Conversione di tipo non valida.
IllegalArgumentException	Argomento di un metodo non valido.
IllegalMonitorStateException	Monitor su thread non valido.
IllegalStateException	Oggetto in uno stato che non consente l'operazione richiesta.
IllegalThreadStateException	Operazione incompatibile con lo stato attuale di un thread.
IndexOutOfBoundsException	Indice non valido.
NegativeArraySizeException	Array creato con dimensione negativa.
NullPointerException	Utilizzo non corretto di un valore null.
NumberFormatException	Conversione non valida di una stringa in un valore numerico.
SecurityException	Violazione delle norme di sicurezza.
StringIndexOutOfBoundsException	Indice non valido per i caratteri di una stringa.
UnsupportedOperationException	Operazione non supportata.

Gestione delle eccezioni

- In Java, le situazioni anomale che si possono verificare a run-time possono essere controllate tramite meccanismi di gestione delle eccezioni
- Esistono classi che descrivono le possibili anomalie
- Ogni volta che la Java Virtual Machine si trova in una situazione anomala;
    1. sospende il programma
    2. crea un oggetto della classe corrispondente all'anomalia che si è verificata
    3. passa il controllo a un gestore di eccezioni (implementato dal programmatore)
    4. se il programmatore non ha previsto nessun gestore, interrompe il programma e stampa il messaggio di errore

Il costrutto try-catch

- Il costrutto try-catch consente di
    - **monitorare** una porzione di programma (all'interno di un metodo)
    - **specificare** cosa fare in caso si verifichi una anomalia (eccezione)

Si usa cosi:

```
// ... blocchi di codice NON monitorati ....

try {
  // ... blocchi di codice monitorati ....
  }
  catch (Exception e) {
  // ... blocchi di codice da eseguire IN CASO DI ECCEZIONE
  }

// ... altri blocchi di codice NON monitorati ....
```

Gestire le eccezioni

- Un costrutto try-catch può gestire più tipi di eccezione contemporaneamente
- I vari gestori (ognuno denotato da un catch) vengono controllati in sequenza
- Viene eseguito (solo) il primo catch che prevede un tipo di eccezione che è supertipo dell'eccezione che si è verificata
- Quindi, è meglio non mettere Exception per prima (verrebbe richiamata in tutti i casi)
- La variabile e è un oggetto che può contenere informazioni utili sull'errore che si è verificato...

```
try {
  //istruzioni da controllare
}
catch (NumberFormatException e) {
  //codice
}
catch (Exception e) {
  //codice
}
```

---

quando definire un gestore di eccezioni

- Per capire quando preoccuparsi di definire un gestore di eccezioni:
    - bisogna avere un'idea di quali sono le eccezioni più comuni e in quali casi si verificano (esperienza)
    - bisogna leggere la documentazione dei metodi di libreria che si utilizzano:

        - la documentazione della classe Scanner spiega che il metodo nextInt() può lanciare l'eccezione InputMismatchException
    - in alcuni casi le eccezioni non vanno gestite: segnalano un errore di programmazione che deve essere corretto!
        - verifica la correttezza dei cicli nel caso di scorrimento di una collezione

---

Il comando throw

- Il meccanismo delle eccezioni può anche essere usato per segnalare situazioni di errore
- Il comando throw consente di lanciare un'eccezione quando si vuole
- Si può usare la classe Exception, una sua sottoclasse già definita, o una sua sottoclasse custom
- **throw** si aspetta di essere seguito da un oggetto, che solitamente è costruito al momento (tramite new)
- Il costruttore di una eccezione prende come parametro (opzionale) una stringa di descrizione

```
throw new Exception("Operazione non consentita");
     throw new AritmeticException ();
     throw new EccezionePersonalizzata ();
```

- Il comando throw si può usare direttamente dentro un try-catch,
- l'uso più comune di throw è all'interno dei metodi
- L'utilizzo di throw dentro a un metodo consente di interrompere il metodo in caso di situazioni anomale:
    - parametri ricevuti errati
    - operazione prevista dal metodo non realizzabile
    - (esempio: prelievo dal conto corrente di una somma superiore al saldo)
- Chi invoca il metodo dovrà preoccuparsi di implementare un gestore delle eccezioni possibilmente sollevate
- Questo consente di evitare valori di ritorno dei metodi che servono solo a dire se l'operazione è andata a buon fine
- in caso di problemi si lancia l'eccezione, non si restituisce un valore particolare

---

parola chiave **throws**

- Un metodo che contiene dei comandi throw deve elencare le eccezioni che possono essere sollevate
- L'elenco deve essere fatto nell'intestazione, usando la parola chiave **throws**
- throws si usa nell'intestazione del metodo
- throw si usa all'interno (nel punto in cui si verifica l'errore) public void preleva(int somma)

```
throws IOException , IllegalParameterException { ... }
```

Terminologia

- **Errore**: problema con la logica applicativa, errore del programmatore (non gestibile)
- **Eccezione**: evento anomalo recuperabile

Una istruzione non terminata

- Causa la creazione di un oggetto che rappresenta quanto è successo
- Tale oggetto appartiene a una classe derivata da Throwable
- Tali oggetti sono le eccezioni

Si dice che l'eccezione

- Viene "gettata" (thrown) e
- In seguito deve essere "gestita" ovvero "catturata" (catch)

Costrutti per la gestione delle eccezioni

- try { } ... catch {}
  - "Getta" l'eccezione a livello di un blocco di istruzioni
  - "Cattura" l'eccezione effettuandone la gestione
- throws
  - "Getta" l'eccezione a livello di metodi
- throw
  - "Getta" l'eccezione a livello di codice / istruzioni

---

try ... catch

- Cattura le eccezioni generate in una regione di codice

```
try {
  // codice in cui possono verificarsi le eccezioni
...
}
catch (IOException e) {
  // codice per gestire IOException e
...
}
```

Per catturare eccezioni di classi diverse si possono usare blocchi catch multipli

```
try {
...
}
catch(MalformedURLException mue) {
// qui recupero l'errore "malformedURL"
...
}
catch(IOException e) {
// qui recupero tutti altri errori di IO
...
}
```

Costrutti try-catch possono essere annidati

(catch che include try-catch)

Il blocco "finally" esegue istruzioni al termine del blocco try-catch

- sia che si verifichino le eccezioni
- sia che NON si verifichino le eccezioni
- Anche in presenza di istruzioni `return, break e continue`

```
try {
...
}
catch (...) {
...
}
catch (...) {
...
}
...
finally {
...
}
```

Permette a un metodo di gettare

eccezioni () throws <classeEccezione 1 > [, <classeEccezione 2 > ... [,]...] { ... }

Le eccezioni gettate sono catturate

(responsabilità) dal chiamante si chiama il metodo leggi deve sapere se la lettura è andata a buon fine oppure no

```
* Con try-catch gestiamo l'eccezione a livello
del chiamato (metodo leggi)
...
byte b[] = new byte[10];
try {
System.in.read (b);
} catch (Exception e) {
...
}
...
```6* Sapere se la lettura è andata a buon fine, non
"interessa" tanto al chiamato (metodo leggi)
quanto al chiamante
static String leggi (String val) throws
IOException {
byte b[] = new byte[10];
System.in.read (b); // Senza try ... Catch
val = "";
for (int i=0; i<b.length; i++) {
val = val + (char) b[i];
}
return (val);
}
```

---

throw

Permette di "gettare" in modo "esplicito" una eccezione a livello di codice `throw <oggettoEccezione>`

Provoca

- L'interruzione dell'esecuzione del metodo
- L'avvio della fase di cattura dell'eccezione generata
- Dato che le eccezioni sono oggetti, chi getta l'eccezione deve creare l'oggetto eccezione (operatore new) che la descrive

```
if ( y==0 ){
throw new ArithmeticException (
"Frazione con denominatore nullo.");
}
z = x/y;
```

Classi di Eccezioni

- È una classe, subclass di Throwable o discendenti, definita in java.lang
- Error: hard failure
```

- Exception: non sistemiche
- RuntimeException: il compilatore non forza il catch
- Error
- Gli errori sono trattabili ma in genere costituiscono situazioni non recuperabili
- OutOfMemoryError
- Exception

Definizione di una eccezione

- È possibile dichiarare eccezioni proprie, se quelle fornite dal sistema (java.lang) non sono sufficienti
- Si realizza creando sottoclassi di Throwable
- Tali sottoclassi sono del tutto "assimilabili" a classi "standard", e.g., possono
  - ereditare attributi e metodi
  - ridefinire il metodo costruttore
  - definire dei metodi get/set
  - etc.

Esempi

EccezioneArray

```
public class EccezioneArray {
  public static void main(String[] args) {
  int[] a = {5,3,6,5,4};
  // attenzione al <=...
  for (int i=0; i<=a.length; i++)
  System.out.println(a[i]);
  System.out.println("Ciao");
  }
}
```

EccezioneAritmetico

```
import java.util.Scanner;
public class EccezioneAritmetico {
  public static void main(String[] args) {
  Scanner input = new Scanner(System.in);
  System.out.println("Inserisci due interi");
  int x = input.nextInt();
  int y = input.nextInt();
  System.out.println(x/y);
  // che succede se y == 0??
```

```
    }
}
```

EccezioneFormato

```
import java.util.Scanner;
public class EccezioneFormato {
  public static void main(String[] args) {
  Scanner input = new Scanner(System.in);
  System.out.println("Inserisci un intero");
  int x = input.nextInt();

  // che succede se l'utente inserisce un carattere?
  System.out.println(x);
  }
}
```

Esempio gestione eccezione: EccezioneAritmetico

```
import java.util.Scanner;
public class EccezioneAritmetico {
  public static void main(String[] args) {
  Scanner input = new Scanner(System.in);

  System.out.println("Inserisci due interi");
  int x = input.nextInt();

  int y = input.nextInt();

  try { System.out.println(x/y);
  System.out.println("CIAO");

  }
  catch (ArithmeticException e) {
  // se si verifica un eccezione di tipo ArithmeticException
  // nella divisione x/y il programma salta qui (non stampa CIAO)
  System.out.println("Non faccio la divisione..."); // gestita l'anomalia,
l'esecuzione riprende...
  }
  System.out.println("Fine Programma"); }
}
```

Esempio gestione eccezione: EccezioneFormato

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class EccezioneFormato {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("Inserisci un intero");
    int x;
    boolean ok;
    do {
      ok = true;
      try {
        x = input.nextInt();
        System.out.println(x);
      }
      catch (InputMismatchException e) {
        input.nextLine(); // annulla l'input ricevuto
System.out.println("Ritenta...");
        ok = false;
      }
    } while (!ok);
  }
}
```

Esempio: controllo correttezza parametri - Rettangolo

```
public class Rettangolo {
  private base;
  private altezza;
  // ... altri metodi e costruttori

  public void setBase(int x) throws EccezioneBaseNegativa {
    if (x<0) throw new EccezioneBaseNegativa()
    else base=x;
  }
}
```

EccezioneBaseNegativa

```
public class EccezioneBaseNegativa extends Exception {
  EccezioneBaseNegativa() {
    super ();
  }

  EccezioneBaseNegativa(String msg) {
    super(msg);
```

```
    }
}
```

System.in.read

- può provocare una eccezione controllata di tipo IOException
- Occorre quindi inserirla in un blocco

```
try...catch...
byte b[] = new byte[10];
try {
System.in.read (b);
} catch (Exception e) {
...
}
```