Gestione degli Errori in Java

Emilio Di Giacomo e Walter Didimo

Errori in fase di esecuzione

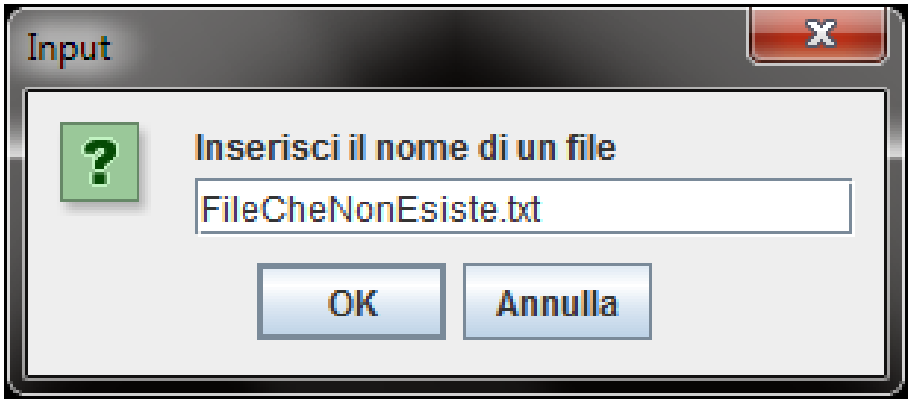
- Un programma può contenere o generare errori in fase di esecuzione, di varia natura:
  - errori di robustezza: dovuti a situazioni limite mal controllate dal programmatore (ad es. si riceve un valore negativo laddove ci si aspettava un valore positivo)
  - errori di natura esterna: un evento esterno imprevisto impedisce la corretta prosecuzione del programma (ad es. si cerca di leggere un file danneggiato)
  - errori di logica: il programma fornisce output inesatti a causa di un errore di logica nell'implementazione di qualche algoritmo da parte del programmatore

Errori in fase di esecuzione

- In linea di principio un programma dovrebbe essere esente da errori di logica e di robustezza
- Essi infatti derivano da un non perfetto lavoro del programmatore
- Gli errori di natura esterna invece dovrebbero essere gestiti dal programmatore
  - al verificarsi di tali errori cioè il programma dovrebbe reagire in maniera controllata

Esempio

- Supponiamo che un programma ci chieda di inserire il nome di un file (ad esempio per leggerlo)
- E supponiamo che inseriamo un nome non esistente

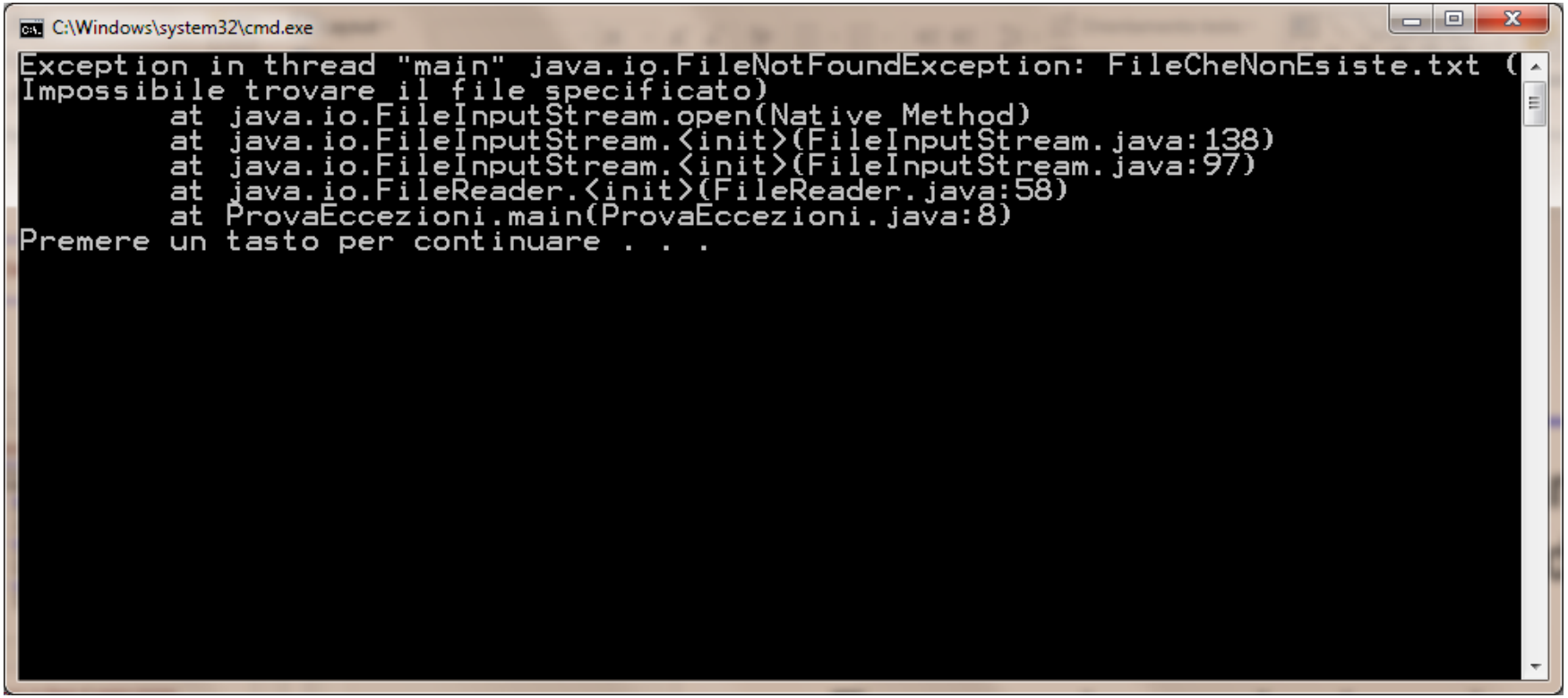


Esempio

- L'inserimento di un nome di file non esistente ovviamente causerà un errore
- Esso può essere gestito...

Esempio

- L'inserimento di un nome di file non esistente ovviamente causerà un errore
- ...oppure no!



Gestione degli errori

- Un modo per gestire errori a tempo di esecuzione prevede di inserire opportune istruzioni di controllo allo scopo di capire se le condizioni necessarie per la corretta esecuzione del programma sono verificate
- Questo approccio è però invasivo
  - esso tende a "sporcare" il codice
  - si perde di vista il flusso principale dell'esecuzione

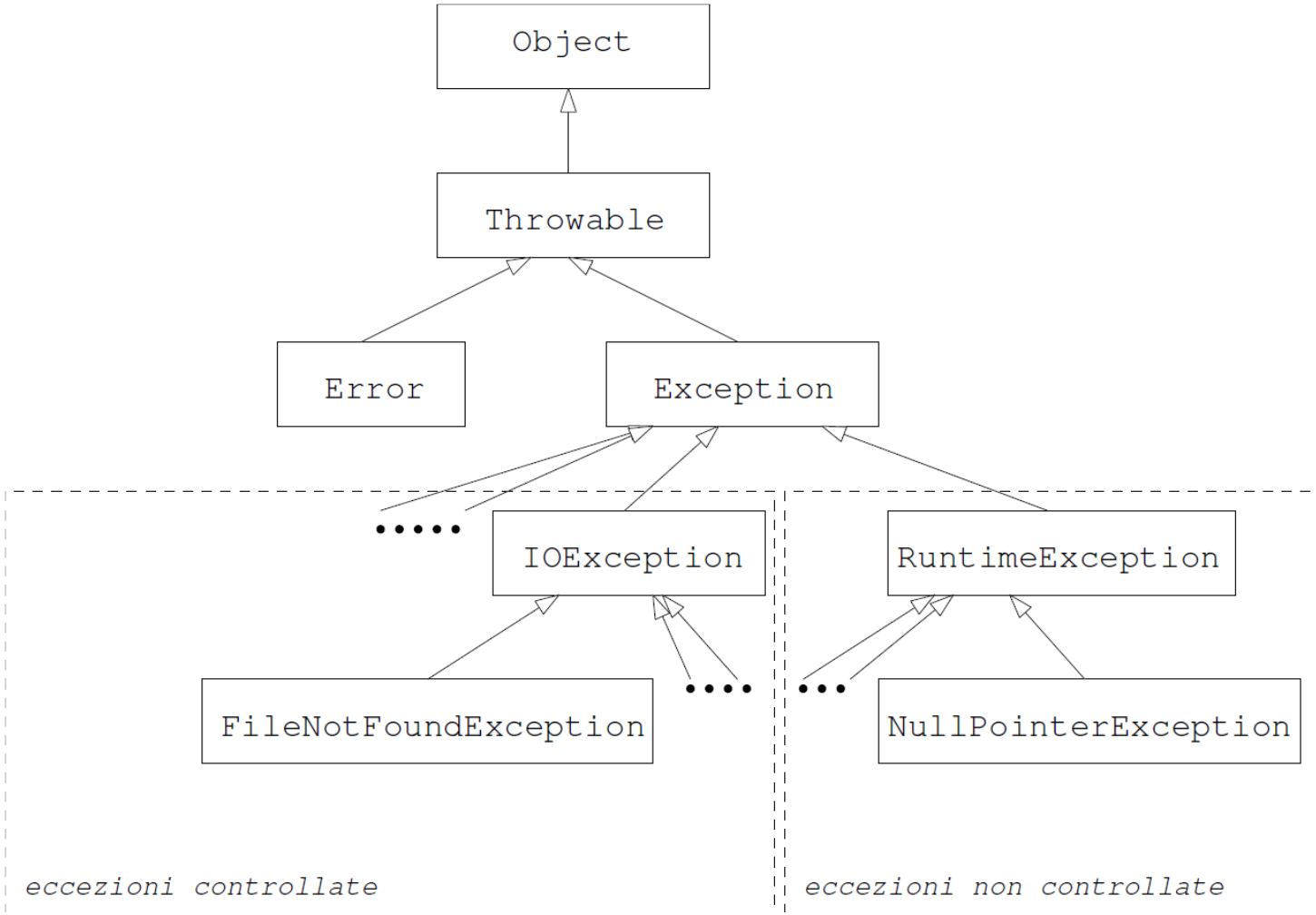
Gestione degli errori

- Java offre un meccanismo efficace e relativamente poco invasivo per gestire gli errori a tempo di esecuzione
- Esso va sotto il nome di gestione delle eccezioni
- Con eccezione si intende una situazione di errore

Eccezioni in Java

- Le eccezioni sono rappresentate tramite oggetti
- Ogni volta che si verifica una situazione di errore viene creato un oggetto che rappresenta l'eccezione che si è avuta
  - in gergo si dice che viene lanciata un'eccezione
- Nell'API Java esiste una gerarchia di classi che modellano molti tipi diversi di eccezioni
- È possibile estendere tale gerarchia aggiungendo classi per modellare nuove tipologie di eccezioni

Gerarchia delle eccezioni



Gerarchia delle eccezioni

- In cima alla gerarchia troviamo la classe *Throwable* che è estesa dalle classi *Error* e *Exception*
- *Error* e le sue sottoclassi rappresentano errori interni della JVM
  - *OutOfMemoryError*, *InternalError*, ecc.
- Sono errori associati a situazioni non recuperabili che causano la terminazione del programma
- Per questo motivo questo tipo di errori non vengono gestiti

Gerarchia delle eccezioni

- *Exception* e le sue sottoclassi rappresentano invece errori che possono essere recuperati o che comunque non causano necessariamente la terminazione del programma
- Le eccezioni sono ulteriormente suddivise in due categorie
  - le eccezioni non controllate: *RuntimeException* e sue sottoclassi
  - le eccezioni controllate: tutte le altre

Gerarchia delle eccezioni

- Le eccezioni non controllate sono tipicamente errori di robustezza o di logica
  - *NullPointerException*, *ArrayIndexOutOfBoundException*
- In linea di principio non dovrebbero esserci
- Non è obbligatorio gestirle, ma lo si può fare
- Le eccezioni controllate sono invece quelle che corrispondono a errori di natura esterna
  - *IOException*, *FileNotFoundException*
- Devono essere gestite

Come si gestiscono le eccezioni

- Immaginiamo che una istruzione *istr* all'interno di un metodo *met* generi un'eccezione
  - indichiamo con *ex* l'eccezione generata e con *ClassEx* la sua classe
- Sono possibili due casi
- Caso 1: Il programmatore non ha definito alcuna politica di gestione di un eccezione di tipo *ClassEx*
  - il programma termina bruscamente
  - questo caso può verificarsi solo per le eccezioni non controllate

Come si gestiscono le eccezioni

- Caso 2: Il programmatore ha definito una politica di gestione di un eccezione di tipo *ClassEx*
  - Ci sono due politiche possibili: rilancio e cattura
- Se l'eccezione viene rilanciata l'esecuzione del metodo *met* viene interrotta e il controllo passa al metodo *metC* che ha invocato *met*
- *metC* si trova adesso nella stessa situazione di *met*:
  - una sua istruzione ha lanciato un'eccezione
  - anche *metC* può gestire o rilanciare l'eccezione
  - se *metC* è il *main* e rilancia l'eccezione il programma termina bruscamente

Come si gestiscono le eccezioni

- Se l'eccezione viene catturata, vuol dire che il programmatore ha scritto un codice per la gestione dell'eccezione
  - l'esecuzione del metodo *met* viene interrotta e si passa ad eseguire il codice di gestione dell'errore

Rilanciare le eccezioni

- Per indicare che un metodo rilancia un'eccezione di un certo tipo *ClassEx* dobbiamo far seguire il suo prototipo dalla clausola *throws ClassEx*
  - tale clausola indica che il metodo può lanciare un'eccezione di tipo *ClassEx*
- Se un metodo può lanciare più tipi di eccezioni questi possono essere elencati dopo la parola chiave *throws* separati da virgole
  - *throws ClassEx1, ClassEx2, ..., ClassExk*

Rilanciare le eccezioni: esempio

- Consideriamo il seguente codice

```
public static void copia(int[] a, int[] b){
      for (int i = 0; i<a.length; i++)
            b[i] = a[i];
}
```

- Qualora la dimensione di *b* fosse minore di quella di *a*, verrebbe generata un'eccezione di tipo *ArrayIndexOutOfBoundException*

Rilanciare le eccezioni: esempio

- Tale eccezione è di tipo non controllato quindi non siamo obbligati a prenderla in considerazione
  - in questo caso ci troveremmo nel Caso 1
- Supponiamo di voler rilanciare l'eccezione in modo da demandarne la gestione al metodo chiamante

```
public static void copia(int[] a, int[] b)
        throws ArrayIndexOutOfBoundsException{
      for (int i = 0; i<a.length; i++)
           b[i] = a[i];
}
```

Rilanciare le eccezioni: esempio

- Il metodo precedente potrebbe generare anche un'eccezione di tipo *NullPointerException*, se ad *a* o *b* venisse passato un riferimento nullo
- Supponiamo di voler rilanciare anche questa eccezione

```
public static void copia(int[] a, int[] b)
        throws ArrayIndexOutOfBoundsException,
                NullPointerException{
        for (int i = 0; i<a.length; i++)
             b[i] = a[i];
}
```

Rilanciare le eccezioni: esempio

- Se un metodo dichiara di lanciare eccezioni di un tipo *ClassEx*, esso può rilanciare eccezioni di quella classe o di una sottoclasse
- Il seguente codice quindi rilancia qualunque tipo di *RuntimeException*

```
public static void copia(int[] a, int[] b)
      throws RuntimeException{
      for (int i = 0; i<a.length; i++)
            b[i] = a[i];
}
```

Rilanciare le eccezioni: commenti

- Se un metodo *met* dichiara di lanciare eccezioni di un tipo *ClassEx* e *ClassEx* è un eccezione controllata allora i metodi che richiamano *met* devono definire una politica di gestione (rilancio o cattura) delle eccezioni di tipo *ClassEx*

Catturare le eccezioni

- Per catturare le eccezioni si usano le clausole *try* e *catch*
- Il codice che può generare eccezioni viene racchiuso in un blocco definito dalla clausola *try* e seguito da un blocco *catch* che contiene il codice da eseguire in caso venga sollevata un'eccezione di un certo tipo

Clausole try e catch

- La sintassi delle clausole *try* e *catch* è la seguente

```
try{
        codice che può generare eccezioni
} catch (<tipo eccezione> <id eccezione>){
        codice da eseguire per gestire un'eccezione
        di tipo <tipo eccezione>
}
```

Clausole try e catch

- *<tipo eccezione>* è il nome di una classe che estende *Throwable*
- *<id eccezione>* è una variabile di tipo *<tipo eccezione>* in cui verrà memorizzato un riferimento all'eccezione generata
  - tale variabile è locale al blocco *catch*

Clausole try e catch

- Un blocco *try* può essere seguito da più blocchi *catch* per gestire eccezioni di tipo diverso

```
try{
        codice che può generare eccezioni
} catch (<tipo eccezione 1> <id eccezione 1>){
        codice da eseguire per gestire un'eccezione
        di tipo <tipo eccezione 1>
} catch (<tipo eccezione 2> <id eccezione 2>){
        codice da eseguire per gestire un'eccezione
        di tipo <tipo eccezione 2>
}
```

Clausole try e catch

- In presenza di un blocco *try-catch* la JVM procede come segue
  - prova ad eseguire il codice del blocco *try*
  - se non vengono generate eccezioni i blocchi *catch* vengono ignorati
  - se invece viene generata un eccezione l'esecuzione del blocco *try* viene interrotta e si analizzano i blocchi *catch* in ordine
  - se ne viene trovato uno per la gestione del tipo di eccezione sollevata viene eseguito il codice di questo blocco
  - i restanti blocchi *catch* vengono ignorati

Catturare le eccezioni: esempio

- Riconsideriamo il seguente codice e supponiamo di voler gestire le eccezioni di tipo *ArrayIndexOutOfBoundException* e *NullPointerException*

```
public static void copia(int[] a, int[] b){
      for (int i = 0; i<a.length; i++)
            b[i] = a[i];
}
```

Catturare le eccezioni: esempio

```
public static void copia(int[] a, int[] b){
        try{
                for (int i=0; i<a.length; i++)
                        b[i]=a[i];
        }catch(ArrayIndexOutOfBoundsException ex){
                System.out.println(ex);
                System.out.println("Array b troppo piccolo");
        }catch(NullPointerException ex){
                System.out.println(ex);
                System.out.println("Uno dei due array è nullo");
        }
}
```

Clausola finally

- Congiuntamente alla clausola *try* (e *catch*) è possibile utilizzare anche la clausola *finally*
- Questa va posta di seguito all'ultimo blocco *catch* o in sostituzione dei blocchi *catch*

```
try{
        codice che può generare eccezioni
} finally{
        blocco finally
}
```

Clausola finally

```
try{
        codice che può generare eccezioni
} catch (<tipo eccezione 1> <id eccezione 1>){
        blocco catch 1
} catch (<tipo eccezione 2> <id eccezione 2>){
        blocco catch 2
} finally{
        blocco finally
}
```

Clausola finally

- Il codice del blocco *finally* viene eseguito in ogni caso dalla JVM
- Se il codice del blocco *try* non genera eccezioni, al termine del blocco *try* viene eseguito il blocco *finally*
- Se il codice del blocco *try* genera un'eccezione, la sua esecuzione si interrompe, viene eseguito il blocco *catch* opportuno e al termine di questo viene eseguito il blocco *finally*

Clausola finally

- Il blocco *finally* viene tipicamente utilizzato per rilasciare risorse che sono state allocate
- Ad esempio, supponiamo che un metodo richieda l'apertura di un file
- Se il metodo termina senza eccezioni, al termine il file deve essere chiuso
- Anche nel caso in cui il metodo generi un'eccezione il file deve essere chiuso
- La chiusura del file può essere quindi gestita nel blocco *finally*

Clausola finally: esempio

- La classe *FileWriter* (del package *java.io*) permette di scrivere file testuali
- Il metodo *write(String s)* di tale classe scrive la stringa *s* sul file
  - tale istruzione genera un'eccezione di tipo *IOException* in caso non sia possibile scrivere sul file per qualunque motivo
- il metodo *close()* chiude il file
  - tale istruzione genera un'eccezione di tipo *IOException* in caso non sia possibile chiudere il file per qualunque motivo

Clausola finally: esempio

- Scriviamo un semplice metodo che riceve un oggetto *FileWriter* e una stringa da scrivere sul file
- Se la scrittura non avviene con successo viene stampato un messaggio di errore
- In ogni caso alla fine viene chiuso il file
- Se la chiusura non va a buon fine il metodo rilancia un'eccezione di tipo *IOException*

Clausola finally: esempio

```
public static void scriviEChiudi(String str, FileWriter f)
                    throws IOException{
    try{
        f.write(str);
    }catch(IOException ex){
        System.out.println("Errore di scrittura nel file");
    }finally{
        f.close();
    }
}
```

Come sollevare eccezioni

- Finora abbiamo visto come trattare eccezioni generate da qualche istruzione
- Ma come si genera un'eccezione per la prima volta?
- L'eccezione è un oggetto e quindi può essere creata come tutti gli oggetti tramite l'operatore *new*
- Per lanciare un'eccezione si usa l'istruzione *throw*
  - se l'eccezione è di tipo controllato allora il metodo che la lancia deve dichiararlo tramite la clausola *throws*

Sollevare eccezioni: esempio

- Supponiamo di voler scrivere un metodo che data la misura g di un angolo in gradi calcoli la misura r in radianti secondo la formula $r = \frac{g \cdot \pi}{180}$
- Se il valore g dato è minore di 0 o maggiore di 360 verrà generata un eccezione di tipo *IllegalArgumentException*

Sollevare eccezioni: esempio

```
public static double convertiInRadianti(double gradi)
        throws IllegalArgumentException{
        if (gradi<0 || gradi>360)
                throw new IllegalArgumentException();
        double radianti = (gradi * Math.PI)/180;
        return radianti;
}
```

- Si noti che in caso venga generata l'eccezione il messaggio che vedremo sarà

```
java.lang.IllegalArgumentException
```

Versione alternativa

```
public static double convertiInRadianti(double gradi)
        throws IllegalArgumentException{
        if (gradi<0 || gradi>360)
                throw new IllegalArgumentException("valore
                        in gradi non compreso tra 0 e 360");
        double radianti = (gradi * Math.PI)/180;
        return radianti;
}
```

- In questo caso il messaggio che vedremo sarà

```
java.lang.IllegalArgumentException: valore
        in gradi non compreso tra 0 e 360
```

Definizione di nuove eccezioni

- Come abbiamo già detto, è possibile definire classi che modellino nuove tipologie di eccezioni
- Supponiamo di voler definire una classe per rappresentare in maniera più specifica il tipo di eccezione che può essere sollevata dal metodo *convertiInRadianti*
- Potremmo chiamare tale classe *InvalidAngleDegreesException*

Definizione di nuove eccezioni

```
public class InvalidAngleDegreesException
             extends IllegalArgumentException{
  /* costruttore senza parametri
  */
  public InvalidAngleDegreesException(){
      super();
  }
  /* costruttore che permette di
     associare un messaggio all'eccezione
  */
  public InvalidAngleDegreesException(String mess){
      super(mess);
  }
}
```

Definizione di nuove eccezioni

```
public static double convertiInRadianti(double gradi)
                        throws InvalidAngleDegreesException{
      if (gradi<0 || gradi>360)
        throw new InvalidAngleDegreesException("valore in
                    gradi non compreso tra 0 e 360!");
      double radianti = (gradi * Math.PI)/180;
      return radianti;
}
```