Programmazione Java
Eccezioni

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Sommario

» Eccezioni
- Introduzione
- Blocco try/catch
- Clausola finally
- Gerarchia eccezioni
  - Checked e unchecked
- Keyword `throws` e `throw`
- Overriding dei metodi
- Creare delle proprie eccezioni
- Stacktrace

Eccezioni

» Momento ideale per individuare errori è durante compilazione

» Non sempre è possibile quindi è necessario un meccanismo durante esecuzione che gestisca errori

» Linguaggi come C generalmente gestiscono gli errori mediante convenzioni

- Valore particolare
- Flag globale che viene esaminato dal destinatario

» **Definizione di eccezione**

- Evento che si verifica durante l'esecuzione di un programma che interrompe il normale flusso delle istruzioni
- Esempi
  - Crash dell'HD
  - Accesso al di fuori dell'array

Eccezioni

- Esempio: Supponiamo di realizzare una funzione che legge un file e lo pone in memoria

- Pseudo-codice

```
readFile {
   open the file;
   determine its size;
   allocate that much memory;
   read the file into memory;
   close the file;
}
```

Eccezioni

» Funzione sembra corretta ma vengono ignorati errori potenziali ovvero

- Cosa accade se il file non può essere aperto?
- Cosa accade se la lunghezza del file non può essere determinata?
- Cosa accade se non vi è abbastanza memoria da allocare?
- Cosa accade se la lettura fallisce?
- Cosa accade se il file non può essere chiuso?

Eccezioni


```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
```

Eccezioni


```
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

Eccezioni

- Si è passati dalle 7 linee originali a 29 (+400%)
- Flusso normale del codice si *perde* all'interno della gestione degli errori rendendo difficile da leggere
- Maggiore difficoltà nella manutenzione del codice
- Soluzione
  - Separazione tra codice normale e condizioni di errore

Eccezioni


```
readFile {
        try {
            open the file;
            determine its size;
            allocate that much memory;
            read the file into memory;
            close the file;
        } catch (fileOpenFailed) {
            doSomething;
        } catch (sizeDeterminationFailed) {
            doSomething;
        } catch (memoryAllocationFailed) {
            doSomething;
        } catch (readFailed) {
            doSomething;
        } catch (fileCloseFailed) {
            doSomething;
        }
}
```

- `readFile()` è un metodo che può generare eccezioni
- In Java è possibile separare l'implementazione della soluzione (corpo del `try`) delegando la gestione degli errori in un'altra posizione del programma (gestori delle eccezioni identificati con la parola chiave `catch`)

Eccezioni

» Passi quando si verifica eccezione

- Creato un oggetto *eccezione* e posto nell'heap
- Flusso di esecuzione corrente interrotto
- Sistema di run-time ha il compito di trovare del codice (tramite lo stack delle chiamate) che gestisce l'eccezione (*exception handler*)
  - Individua il corretto exception handler utilizzando il tipo dell'eccezione sollevata
  - Se non è presente alcun exception handler il sistema di runtime termina il programma

Eccezioni > try/catch

- Si tenta di eseguire il codice e si intercetta un'eccezione si pone rimedio
- Sintassi

```
try {
    Istruzioni
}
catch (<Tipo eccezione> <Identificatore>) {
    Altre istruzioni
}
```

- L'istruzione `try` identifica un blocco d'istruzioni in cui può verificarsi un'eccezione

Eccezioni > try/catch

- Un blocco `try` è seguito da una o più clausole `catch`, che specificano quali eccezioni vengono gestite
- Ogni clausola `catch` corrisponde a un tipo di eccezione sollevata
- Quando si verifica un'eccezione, la computazione continua con la prima clausola che corrisponde all'eccezione sollevata
- Al termine dell'esecuzione della clausola catch trovata, l'eccezione è considerata interamente gestita (la ricerca di gestori si interrompe, diversamente dall'istruzione switch)

Eccezioni > try/catch

» Esempio

```
public class TestWithoutException {

   public static void main(String[] args) {
      for (int i= 0; i < 10; i++) {
               System.out.println("Argomento i(" + i + "): "
                                  + args[ i ]);
      }
   }
}
```

(Provare a lanciare TestWithoutException.java senza nessun argomento)

Eccezioni > try/catch

» Esempio con eccezione

```
public class TestWithException {

   public static void main(String[] args) {
      try {
        for (int i= 0; i < 10; i++) {
           System.out.println("Argomento i-esimo(" + i +
                              "): " + args[ i ]);
        }
      }
      catch(ArrayIndexOutOfBoundsException e ) {
         System.out.println("Si e' verificata l'eccezione");
      }
   }
}
```

(Provare a lanciare TestWithException.java senza nessun argomento)

Eccezioni > try/catch

» Esempio 2

```
public class Test {

   public static void main(String[] args) {

      try {
         anotherMethod(args);
      }
      catch(ArrayIndexOutOfBoundsException e ) {
         System.out.println("Si e' verificata l'eccezione");
      }
   }

   public static void anotherMethod(String[] args) {
       for (int i= 0; i < 10; i++) {
           System.out.println("Argomento i-esimo(" + i + "): " + args[ i ]);
       }
   }
}
```

(Provare a lanciare Test2WithException.java senza nessun argomento)

Eccezioni > finally

- Istruzione `try` può avere una clausola `finally` opzionale
- Se non viene sollevata nessuna eccezione, le istruzioni nella clausola `finally` vengono eseguite dopo che si è concluso il blocco `try`
- Se si verifica un'eccezione, le istruzioni nella clausola `finally` vengono eseguite dopo le istruzioni della clausola `catch` appropriata
- In definitiva se è presente clausola `finally` viene sempre eseguita indipendentemente dal verificarsi o meno di un'eccezione
- Generalmente viene utilizzato per liberare risorse utilizzate all'interno del blocco `try` (es. Files, DB)

Eccezioni > finally


```
try {
  // Regione sorvegliata: attività pericolose
  // che potrebbero sollevare le eccezioni ti dipo A, B, o C
} catch (A a1) {
  // Gestore per la situazione A
} catch (B b1) {
  // Gestore per la situazione B
} catch (C c1) {
  // Gestore per la situazione C
} finally {
  // Attività che vengono comunque eseguite
}
```

Eccezioni > finally

» Esempio

```
public class Test {

   public static void main(String[] args) {
      try {
        for (int i= 0; i < 10; i++) {
           System.out.println("Argomento i-esimo(" + i +
                              "): " + args[ i ]);
        }
      }
      catch(ArrayIndexOutOfBoundsException e ) {
         System.out.println("Si e' verificata l'eccezione");
      } finally {
         System.out.println("Blocco sempre eseguito");
      }
   }
}
```

Eccezioni > Gerarchia

- In Java un oggetto eccezione è sempre un'istanza di una classe derivata da `Throwable`
- Gerarchia si suddivide in due categorie
  - **Error**
    - Errori che si verificano all'interno della VM
      - dynamic linking
      - hard failure
    - Difficilmente è possibile recuperare da errori di questo tipo
    - Esempi
      - `OutOfMemoryError`
      - `StackOverflowError`

Eccezioni > Gerarchia

- **Exception**
  - `RuntimeException` e sue sottoclassi
    - Si verificano quando è stato commesso un errore di programmazione
    - Sono dette **unchecked** (non verificate) e non è obbligatorio gestirle
    - Esempi
      - Cast definito male: `ClassCastException`
      - Accesso ad un puntatore nullo: `NullPointerException`
  - Altre classi che **non** derivano da `RuntimeException`
    - Si verificano quando si è verificato qualcosa di imprevisto
    - Sono dette **checked** (verificate) ed è obbligatorio gestirle ovvero è necessario inserirle in un blocco `try/catch` oppure usare clausola `throws` in un metodo
    - Esempio
      - Apertura di un file: `FileNotFoundException`

Eccezioni > Gerarchia

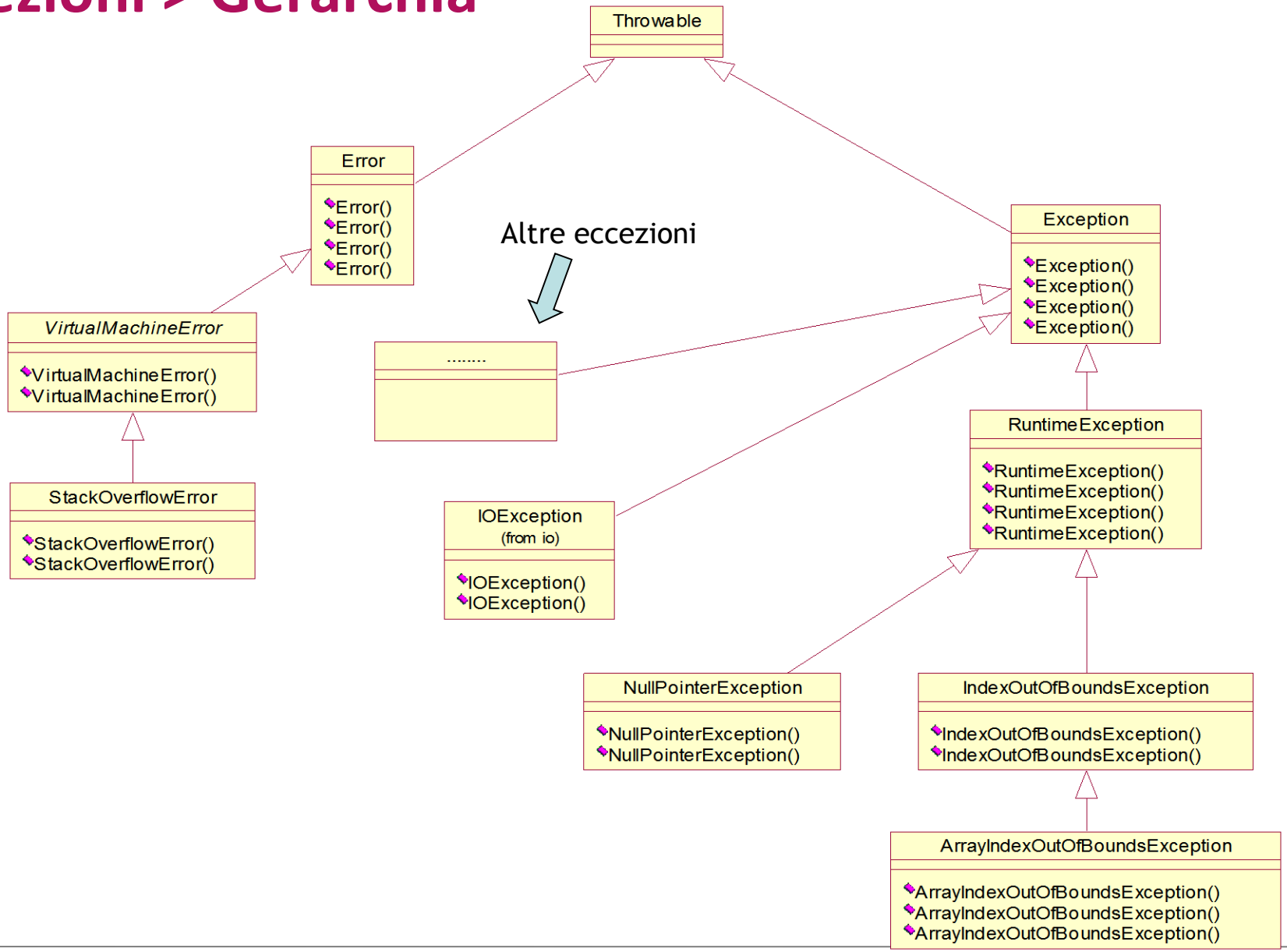
» Per *java.lang.RuntimeException* e *java.lang.Error* e per tutte le class che estendono queste class non c'è bisogno di verificare le eccezioni ma si può comunque usare il try. Subclass di *RuntimeException*

- *ArithmeticException*: ex 3/0
- *ClassCastException*: ex (Object)0
- *IndexOutOfBoundsException*: ex (new int[3])[5]
- *ArrayStoreException*: ex Object x[] = new String[3]; x[0] = new Integer(0);
- *NullPointerException*: ex null.length()

» Subclass di *Error*

- OutOfMemoryError
- StackOverflowError
- NoClassDefFoundError

Eccezioni > Gerarchia



Eccezioni > Gerarchia

» Esempio 1

```
public class TestStackOverflow {
    static int numeroChiamata;

    public static void main(String[] args) {
        funzioneRicorsiva();
    }

    public static void funzioneRicorsiva() {
        System.out.println("Invocazione metodo numero: " +
                           numeroChiamata++);
        funzioneRicorsiva();
    }
}
```

(Vedere TestStackOverflow.java)

Eccezioni > Gerarchia

» Esempio 2

```
class Point {
    int x,y;
    public Point(int x, int y) {
          this.x = x;
          this.y = y;
    }

}

class TestNullPointerException {

    public static void main(String[] args) {
          Point p = null;
          System.out.println("Accesso variabile d'istanza x di p: "
                              + p.x);
    }

}
```

(Vedere TestNullPointerException.java)

Eccezioni > Gerarchia

» Esempio 3

```
import java.io.*;
class TestCheckedException {
    public static void main(String[] args) {
         if (args.length!=1)
             return;
         FileReader reader = null;
         try {
               reader = new FileReader( args[ 0 ] );
         }
         catch(FileNotFoundException e) {
            System.out.println("File non trovato");
         }
         finally {
               if (reader!=null) {
                  try {
                       reader.close();
                  }
                  catch (IOException e) { //Do nothing }
               }
         }
    }
}
```

(Vedere TestCheckedException.java)

Eccezioni > Gerarchia

» Esempio 4

```
import java.io.*;
public class TestMultiCheckedException {
    public static void main( String[] args ) {
        if ( args.length != 1 )
            return;

        BufferedReader reader = null;
        try {
          reader = new BufferedReader( new FileReader( args[ 0 ] ));
          String linea = null;
          while ( ( linea = reader.readLine() ) != null ) {
                System.out.println( "linea letta = " + linea );

          }
        }
        catch ( FileNotFoundException e ) {
            System.out.println( "File non trovato!" );
        }
```

(Vedere TestMultiCheckedException.java)

Eccezioni > Gerarchia


```
        catch ( IOException e ) {
           System.out.println( "Eccezione in lettura!" );
        }
        finally {
             if ( reader != null ) {
                 try {
                     reader.close();
                 }
                 catch ( IOException e ) {
                     //Do nothing
                 }
             }
         }
    }
}
```

(Vedere TestMultiCheckedException.java)

Eccezioni > throws

» Eccezioni checked possono non essere racchiuse all'interno di un blocco `try/catch`

» Si utilizza clausola `throws` all'interno della dichiarazione di un metodo

» Sintassi

```
[modificatoriAccesso]
     [ static | abstract | final | native | synchronized ]*
          tipoRitorno nomeMetodo(listaparametri)
                         [ throws exceptions]
```

Nota: exceptions è una lista di eccezioni separata da "`,`"

```
Es. void f() throws TooBig, TooSmall, DivZero {…}
```

» E' il chiamante che deve racchiudere invocazione metodo all'interno di un blocco `try/catch`

» E' possibile catturare l'eccezione e *rilanciarla* mediante clausola `throw`

– in tal caso è possibile rigenerare un'eccezione diversa da quella intercettata

Eccezioni > throws

» Se invece della seguente dichiarazione

```
void f() throws TooBig, TooSmall, DivZero {…}
```

si usa questa

```
void f() {…}
```

significa che il metodo non genere alcuna eccezione (fatta eccezione delle RuntimeException che possono essere generate in qualsiasi posizione senza bisogno di alcuna specifica)

Eccezioni > throws

- Se il codice del metodo f() può sollevare eccezioni, ma non le gestisce, il compilatore lo rileverà e chiederà di gestirla
- E' possibile "mentire" al compilatore dichiarando che un metodo può generare un'eccezione che in realtà non si può verificare

Eccezioni > throws

» Esempio 1

```
import java.io.FileNotFoundException;

public class TestThrowsKeyword {

    public static void main( String[] args ) {
        if ( args.length != 1 )
            return;

        try {
            readFile( args[ 0 ] );
        }
        catch( FileNotFoundException e  ) {
            System.out.println("File Not Found!");
        }
        catch ( IOException e ) {
            System.out.println("Errore nel file!");
        }
    }
```

(Vedere TestThrowsKeyword.java)

Eccezioni > throws


```
.....................................................
    private static void readFile( String filename )
                        throws FileNotFoundException, IOException {
        BufferedReader reader = new BufferedReader(
                                        new FileReader( filename ) );
        String linea = null;
        while ( ( linea = reader.readLine() ) != null ) {
            System.out.println( "linea letta = " + linea );

        }
        if ( reader != null ) {
            try {
                reader.close();
            }
            catch ( IOException e ) {
                //Do nothing
            }
        }

    }
}
```

(VedereTestThrowsKeyword.java)

Eccezioni > throws

» Esempio 2

```
public class TestThrowKeyword {
    public static void main( String[] args ) {
        if ( args.length != 1 )
            return;

        try {
            readFile( args[ 0 ] );
        }
        catch( FileNotFoundException e  ) {
            System.out.println("File Not Found nel main!");
        }
        catch ( IOException e ) {
            System.out.println("Errore nel file nel main!");
        }
    }
```

……

(Vedere TestThrowKeyword.java, Rethrowing.java, RethrowNew.java)

Eccezioni > throws


```
.....................................
     private static void readFile( String filename )
                throws FileNotFoundException, IOException {
          BufferedReader reader = null;
          try {
             reader = new BufferedReader(
                                  new FileReader( filename ) );
             String linea = null;
             while ( ( linea = reader.readLine() ) != null ) {
               System.out.println( "linea letta = " + linea );
             }
          }
.....................................
```

(VedereTestThrowKeyword.java, Rethrowing.java, RethrowNew.java)

Eccezioni > throws


```
.....................................................
    catch( FileNotFoundException e  ) {
            System.out.println("File Not Found!");
            throw e;
        }
    catch ( IOException e ) {
            System.out.println("Errore nel file!");
            throw e;
        }
    }
    finally {
           if ( reader != null ) {
                try {
                    reader.close();
                }
                catch ( IOException e ) {
                   //Do nothing
                }
            }
    }
}
```

(VedereTestThrowKeyword.java, Rethrowing.java, RethrowNew.java)

Eccezioni > ancora sul finally

- Vedere
  - OnOffSwitch.java
  - WithFinally.java
  - WithFinally2.java
  - AlwaysFinally.java
  - FinallyWorks.java

Eccezioni > criteri di corrispondenza


```
class Annoyance extends Exception {}
class Sneeze extends Annoyance {}

public class Human {

  public static void main(String[] args) {
    try {
      throw new Sneeze();
    } catch(Sneeze s) {
      System.err.println("Caught Sneeze");
    } catch(Annoyance a) {
      System.err.println("Caught Annoyance");
    }

  }
}
```

(Vedere Human.java)

- L'eccezione `Sneeze()` verrà intercettata dalla prima clausola
- Se la prima clausola venisse rimossa lasciando solo `catch(Annoyance a)` Il codice continuerebbe a funzionare
- Se spostassimo la clausola per Sneeze dopo quella di Annoyance, il compilatore darebbe errore

Eccezioni > Metodi di Throwable

» Per avere informazioni specifiche di un'eccezione è possibile invocare i metodi del tipo base Throwable:

- `String getMessage()`

accede al messaggio contenente i dettagli dell'eccezione

- `toString()`

restituisce una breve descrizione dell'oggetto Throwable, compresi i dettagli dell'eccezione, se esistenti

Eccezioni > Metodi di Throwable


```
public class ExceptionMethods {

  public static void main(String[] args) {
    try {
      throw new Exception("My Exception");
    } catch(Exception e) {
      System.err.println("Caught Exception");
      System.err.println("getMessage():" + e.getMessage());
      System.err.println("toString():" + e);
      System.err.println("printStackTrace():");
      e.printStackTrace();
    }

  }
}
```

(Vedere ExceptionMethods.java)

Eccezioni > Metodi di Throwable > printStackTrace

» E' possibile stampare sullo standard di error l'eccezione con il relativo stack delle chiamate

- `public void printStackTrace()`

» Esempio

```
class StackTrace{
     public static void main(String[] args) {
          crunch(null);
     }
     static void crunch(int[] a) {
          mash(a);
     }
     static void mash(int[] b) {
          try {
             System.out.println(b[0]);
          }
          catch(Exception e) {e.printStackTrace();}
     }
 }
```

(Vedere StackTrace.java)

Eccezioni > Metodi di Throwable > getCause

- A partire da Java 1.4 è possibile impostare l'eccezione originale come "causa" della nuova eccezione ovvero si possono annidare
- Eccezioni standard hanno un costruttore con un parametro di tipo `Throwable` che identifica la causa
  - `public Throwable(String message, Throwable cause)`
- E' possibile recuperare la causa originale mediante metodo `getCause`
- E' possibile utilizzare metodo `initCause` stesso effetto del costruttore (vedere DynamicFieldsException.java)

Eccezioni > Metodi di Throwable > getCause

» Esempio

```
class StackTrace2{
    public static void main( String[] args ) {
        crunch( null );
    }

    static void crunch( int[] a ) {
        try {
            mash( a );
        }
        catch ( MyException e ) {
            e.printStackTrace();
            System.err.println( "----------------------" );
            e.getCause().printStackTrace();
            System.err.println( "----------------------" );
        }
    }
```

.....

(Vedere StackTrace2.java)

Eccezioni > Metodi di Throwable > getCause

.....

```
    static void mash( int[] b ) {
        try {
            System.out.println( b[ 0 ] );
        }
        catch ( Exception e ) {
            throw new MyException( "Errore", e );
        }
    }
}
```

(Vedere StackTrace2.java, Cleanup.java, Interface3.java)

Eccezioni > proprie eccezioni

- E’ possibile creare delle proprie eccezioni per indicare condizioni di errore o eccezioni non previste dalla libreria standard di java
- E’ sufficiente derivare da `Exception` o `RuntimeException` o da qualsiasi altra eccezione

Eccezioni > proprie eccezioni


```
class SimpleException extends Exception {}

    public class SimpleExceptionDemo {

      public void f() throws SimpleException {
        System.out.println("Throw SimpleException from f()");
        throw new SimpleException();
      }
      public static void main(String[] args) {
        SimpleExceptionDemo sed = new SimpleExceptionDemo();
        try {
          sed.f();
        } catch(SimpleException e) {
          System.err.println("Caught it!");
        }

      }
}
```

(Vedere SimpleExceptionDemo.java)

Eccezioni > proprie eccezioni


```
class MyException extends Exception {
  public MyException() {}
  public MyException(String msg) { super(msg); }
}

public class FullConstructors {

  public static void f() throws MyException {
    System.out.println("Throwing MyException from f()");
    throw new MyException();
  }
  public static void g() throws MyException {
    System.out.println("Throwing MyException from g()");
    throw new MyException("Originated in g()");
  }
  public static void main(String[] args) {
    try {
      f();
    } catch(MyException e) {
      e.printStackTrace();
    }
    try {
      g();
    } catch(MyException e) {
      e.printStackTrace();
    }
  }

}
```

Fa parte dell'interfaccia `Throwable` e l'effetto è quello di produrre le informazioni sulla sequenza di metodi che sono stati chiamati fino al punto in cui si è generata l'eccezione

(Vedere FullConstructors.java)

Eccezioni > proprie eccezioni


```
class MyException2 extends Exception {
  private int x;
  public MyException2() {}
  public MyException2(String msg) { super(msg); }
  public MyException2(String msg, int x) {
    super(msg);
    this.x = x;
  }
  public int val() { return x; }
  public String getMessage() {
    return "Detail Message: "+ x + " "+ super.getMessage();
  }
}

...
```

(Vedere ExtraFeatures.java)

Eccezioni > proprie eccezioni


```
public class ExtraFeatures {

  public static void f() throws MyException2 {
    System.out.println("Throwing MyException2 from f()");
    throw new MyException2();
  }
  public static void g() throws MyException2 {
    System.out.println("Throwing MyException2 from g()");
    throw new MyException2("Originated in g()");
  }
  public static void h() throws MyException2 {
    System.out.println("Throwing MyException2 from h()");
    throw new MyException2("Originated in h()", 47);
  }
  …
```

(Vedere ExtraFeatures.java)

Eccezioni > proprie eccezioni


```
…
public static void main(String[] args) {
    try {
      f();
    } catch(MyException2 e) {
      e.printStackTrace();
    }
    try {
      g();
    } catch(MyException2 e) {
      e.printStackTrace();
    }
    try {
      h();
    } catch(MyException2 e) {
      e.printStackTrace();
      System.err.println("e.val() = " + e.val());
    }

  }
}
```

(Vedere ExtraFeatures.java)

Eccezioni > proprie eccezioni


```
public class MyException extends RuntimeException {

    public MyException() {
    }

    public MyException( String message ) {
        super( message );
    }

    public MyException( String message, Throwable cause ) {
        super( message, cause );
    }

    public MyException( Throwable cause ) {
        super( cause );
    }
}
```

(Vedere TestMyException3.java)

Eccezioni > proprie eccezioni


```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class TestMyException {
    public static void main( String[] args ) {
        if ( args.length != 1 )
            return;

        try {
            readFile( args[ 0 ] );
        }
        catch ( MyException e ) {
            System.out.println( "MyException!" );
        }
    }
……
```

(Vedere TestMyException3.java)

Eccezioni > proprie eccezioni


```
.....

 private static void readFile( String filename ) {

        BufferedReader reader = null;
        try {
          reader = new BufferedReader( new FileReader( filename ) );
          String linea = null;
          while ( ( linea = reader.readLine() ) != null ) {
              System.out.println( "linea letta = " + linea );

          }
        }
        catch ( FileNotFoundException e ) {
          System.out.println( "File Not Found!" );
          throw new MyException( "MyException", e );
        }

.....
```

(Vedere TestMyException3.java)

Eccezioni > proprie eccezioni


```
        catch ( IOException e ) {
            System.out.println( "Errore nel file!" );
            throw new MyException( "MyException", e );
        }
        finally {
            if ( reader != null ) {
                try {
                    reader.close();
                }
                catch ( IOException e ) {
                    //Do nothing
                }
            }
        }
    }
}
```

(Vedere TestMyException3.java)

Eccezioni > overriding metodi

- Quando si *override* un metodo in una sottoclasse, si è vincolati a poter generare soltanto le eccezioni che sono state specificate nella versione del metodo della classe base
  - Questo assicura che il codice che funziona con la classe base automaticamente funziona con qualsiasi oggetto derivato dalla classe base
- Questa restrizione non si applica ai costruttori delle sottoclassi, che possono lanciare le eccezioni che vogliono
  - L'unica nota è che, dato che il costruttore di una sottoclasse automaticamente invoca il costruttore della superclasse, tutte le eccezioni presenti nella superclasse devono essere specificate anche nel costruttore della sottoclasse
    - Se ne possono aggiungere altre, a differenza dei metodi normali con overriding

Eccezioni > overriding metodi

- Eccezioni verificate inserite in dichiarazione di metodo hanno impatto nell'overriding
- Metodo che effettua l'overriding non può lanciare un'eccezione più generale di quella del metodo della superclasse
- Stessa cosa vale per l'implementazione di metodi dichiarati in un'interfaccia

Eccezioni > overriding metodi

» Esempio 1

```
import java.io.IOException;
public class Base {

    public void metodoA() throws IOException {}
}

import java.io.FileNotFoundException;
public class Derivata extends Base {

    public void metodoA() throws FileNotFoundException { }
}

public class AltraDerivata extends Base {
    public void metodoA() throws Exception { } //ERRORE
}
```

Eccezioni > overriding metodi

» Esempio 2

```
import java.io.IOException;
public interface Interfaccia {
    void metodoA() throws IOException;
}

import java.io.IOException;
public class InterfacciaImpl implements Interfaccia {
    public void metodoA() throws IOException { }
}

public class AltraInterfacciaImpl implements Interfaccia {
    public void metodoA() throws Exception { } //ERRORE
}
```

Eccezioni > overriding metodi

- Esempio 3
  - (vedere ExceptionsInHierarchy.java)

Eccezioni > overriding metodi

» Esempio 4

(vedere StormyInning.java)

Nota:

- Sebbene le specifiche delle eccezioni vengano applicate dal compilatore durante l'ereditarietà, le specifiche delle eccezioni non fanno parte del tipo di un metodo
- Non è possibile definire versioni *sovraccariche* di metodi basati solo su diverse specifiche delle eccezioni