

Gestire le eccezioni

Lezione XI

Scopo della lezione

- Studiare il meccanismo di gestione delle eccezioni
- Implementare nuove eccezioni

Tipi di errori

- Errori rilevati in fase di compilazione: errori di sintassi, ...
- Errori rilevati in fase di esecuzione:
 - dall'utente (errori logici)
 - dalla JVM (eccezioni)

Eccezioni

- Si definisce eccezione un errore la cui localizzazione
 - nello spazio (i.e. all'interno del codice sorgente), e/o
 - nel tempo (rispetto all'istante di esecuzione)risulta a priori difficile quando non impossibile

Esempio

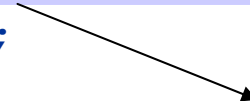
```
import java.io.*;
import prog.io.*;
class EccezioneFileErrata {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        BufferedReader in =
            new BufferedReader(new FileReader("elenco.txt"));
        String line = in.readLine();
        int i=1;
        while(line!=null) {
            i++;
            line = in.readLine();
        }
        video.println("Il file e' lungo " + (i-1) + " righe.");
    }
}
```

Esempio

```
malchiod% javac EccezioneFileErrata.java
EccezioneFileErrata.java:8: unreported exception
    java.io.FileNotFoundException; must be caught or
    declared to be thrown
        new BufferedReader(new FileReader("elenco.txt"));
                ^
EccezioneFileErrata.java:9: unreported exception
    java.io.IOException; must be caught or declared to be
    thrown
    String line = in.readLine();
                ^
EccezioneFileErrata.java:13: unreported exception
    java.io.IOException; must be caught or declared to be
    thrown
        line = in.readLine();
                ^
3 errors
```

Esempio

```
import java.io.*;
import prog.io.*;
class EccezioneFileErrata {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        BufferedReader in =
            new BufferedReader(new FileReader("elenco.txt"));
        String line = in.readLine();
        int i=1;
        while(line!=null) {
            i++;
            line = in.readLine();
        }
        video.println("Il file e' lungo " + (i-1) + " righe.");
    }
}
```



**Cosa succede se
elenco.txt non esiste?**

Eccezioni

- Nel caso si verifichi una **condizione inattesa** (come la mancanza del file nell'esempio precedente), una classe Java lancia un'**eccezione**
- Il compilatore rileva le situazioni che potrebbero generare delle eccezioni e richiede al programmatore di trattarle in modo **dedicato ed esplicito**

Quindi...

```
import java.io.*;
import prog.io.*;
class EccezioneFileErrata {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        BufferedReader in =
            new BufferedReader(new FileReader("elenco.txt"));
        String line = in.readLine();
        int i=1;
        while(line!=null) {
            i++;
            line = in.readLine();
        }
        video.println("Il file e' lungo " + (i-1) + " righe.");
    }
}
```

se elenco.txt non esiste
viene lanciata l'eccezione
FileNotFoundException

Analogamente...

```
import java.io.*;
import prog.io.*;
class EccezioneFileErrata {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        BufferedReader in =
            new BufferedReader(new FileReader("elenco.txt"));
        String line = in.readLine();
        int i=1;
        while(line!=null) {
            i++;
            line = in.readLine();
        }
        video.println("Il file e' lungo " + i + " righe.");
    }
}
```

Queste istruzioni possono
lanciare un'eccezione
IOException

Eccezioni

- La filosofia di base di Java prevede che un codice mal progettato non verrà mai eseguito
- Ogni volta che del codice potrebbe lanciare delle eccezioni, alternativamente
 - deve essere scritto del codice aggiuntivo per gestire le condizioni eccezionali
 - deve essere esplicitamente dichiarata la possibilità di emissione di un'eccezione

Dichiarare le eccezioni

- Quando un metodo contiene codice che potrebbe lanciare delle eccezioni, la relativa intestazione viene modificata indicando quante e quali sono queste eccezioni
- La parola chiave `throws`, inserita dopo l'elenco dei parametri formali nella dichiarazione di un metodo, dichiara quali eccezioni possono essere lanciate

Dichiarare le eccezioni

```
import java.io.*; import prog.io.*;
class EccezioneFile {
    public static void main(String[] args)
        throws FileNotFoundException, IOException {
        ConsoleOutputManager video=new
            ConsoleOutputManager();
        BufferedReader in =
            new BufferedReader(new
                FileReader("elenco.txt"));
        String line = in.readLine();
        int i=1;
        while(line!=null) {
            i++;
            line = in.readLine();
        }
        video.println("Il file e' lungo " + (i-1) + "
            righe.");
    }
}
```

Dichiarare le eccezioni

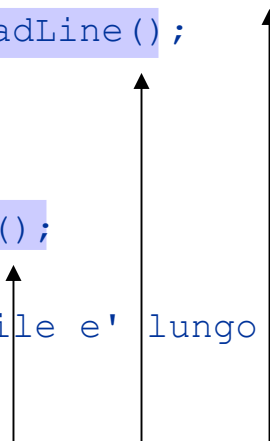
```
malchiod% cat elenco.txt
riga 1
riga 2
riga 3
riga 4
riga 5
riga 6
malchiod% java EccezioneFile
Il file e' lungo 6 righe.
malchiod% rm elenco.txt
malchiod% java EccezioneFile
Exception in thread "main" java.io.FileNotFoundException:
elenco.txt (No such file or directory)
  at java.io.FileInputStream.open(Native Method)
  at java.io.FileInputStream.<init>(FileInputStream.java:103)
  at java.io.FileInputStream.<init>(FileInputStream.java:66)
  at java.io.FileReader.<init>(FileReader.java:41)
  at EccezioneFile.main(EccezioneFile.java:7)
```

Risolvere le eccezioni

- In un generico linguaggio di programmazione gli approcci alla risoluzione delle eccezioni sono principalmente due
 - il programmatore deve verificare che **prima** di eseguire un'istruzione questa non dia luogo ad un'eccezione
 - il programmatore deve occuparsi di gestire le eccezioni solo **dopo** che queste sono state lanciate

Verifica delle eccezioni

```
import java.io.*;
import prog.io.*;
class EccezioneFileErrata {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        BufferedReader in =
            new BufferedReader(new FileReader("elenco.txt"));
        String line = in.readLine();
        int i=1;
        while(line!=null) {
            i++;
            line = in.readLine();
        }
        video.println("Il file e' lungo " + (i-1) + " righe.");
    }
}
```



Nel primo approccio, prima di eseguire ognuna di queste istruzioni devo verificarne la validità

Gestire le eccezioni

Java adotta la seconda soluzione, in quanto

- permette spesso di scrivere un unico blocco di codice che gestisca eccezioni che si possono presentare in parti diverse del codice
- lascia che il programmatore possa separare l'implementazione del codice che gestisce le situazioni tipiche da quello che gestisce le situazioni eccezionali

Gestire le eccezioni

La gestione delle eccezioni avviene

- dichiarando all'interno di un metodo una **sezione critica** di codice che potrebbe lanciare una o più eccezioni
- facendo seguire questa sezione da uno o più blocchi di codice deputati a gestire le eccezioni che possono essere lanciate

Gestire le eccezioni

- Una sezione critica viene indicata racchiudendo il codice relativo in una coppia di parentesi graffe precedute dalla parola chiave `try`
- La parte di codice che gestisce un'eccezione segue questo blocco, racchiusa tra parentesi graffe e preceduta dalla parola chiave `catch` che specifica anche l'eccezione gestita

Gestione delle eccezioni

```
try {
```

Sezione critica

```
} catch (Exception_1 e) {
```

**Gestione dell'eccezione
di tipo Exception_1**

```
}  
catch (Exception_2 e) {
```

**Gestione dell'eccezione
di tipo Exception_2**

```
}
```

- Se durante l'esecuzione della sezione critica non vengono lanciate eccezioni, l'esecuzione procede in modo regolare e i blocchi individuati da catch vengono ignorati

Gestione delle eccezioni

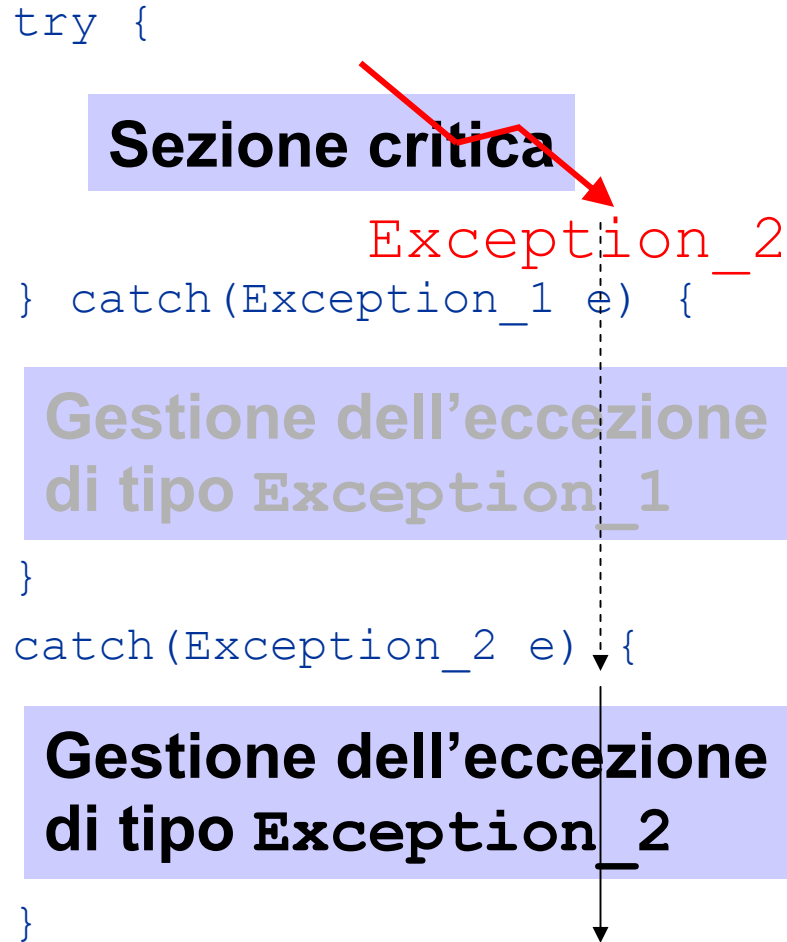
```
try {  
    Sezione critica  
} catch (Exception_1 e) {  
    Gestione dell'eccezione  
di tipo Exception_1  
}  
catch (Exception_2 e) {  
    Gestione dell'eccezione  
di tipo Exception_2  
}
```

The diagram illustrates the execution flow of an exception handler. It starts with a red arrow pointing from the 'Sezione critica' (critical section) to the 'Exception_1' label. A dashed arrow then points from 'Exception_1' to the first catch block. Another dashed arrow points from the first catch block to the second catch block. A final dashed arrow points from the second catch block downwards, indicating the end of the exception handling process.

- Se viene lanciata un'eccezione di tipo `Exception_1`
 - viene interrotta l'esecuzione della sezione critica
 - viene eseguito il blocco catch corrispondente
 - viene ignorato il rimanente blocco catch

Gestione delle eccezioni

```
try {  
    Sezione critica  
} catch (Exception_1 e) {  
    Gestione dell'eccezione  
di tipo Exception_1  
}  
catch (Exception_2 e) {  
    Gestione dell'eccezione  
di tipo Exception_2  
}
```



- Se viene lanciata un'eccezione di tipo `Exception_2`
 - viene interrotta l'esecuzione della sezione critica
 - viene ignorato il primo blocco catch
 - viene eseguito il blocco catch corrispondente a `Exception_2`

Eccezioni vs. `switch`

- La valutazione delle clausole `catch` nella gestione delle eccezioni ricorda il costruito `switch`
- Viene eseguita solo la porzione di codice corrispondente all'eccezione lanciata, **MA**
- Non sono previste istruzioni speciali (come `break`) per chiudere i singoli blocchi `catch`

Esempio

```
import java.io.*;
import prog.io.*;

class EccezioneGestita {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        try {
            BufferedReader in =
                new BufferedReader(new FileReader("elenco.txt"));
            String line = in.readLine();
            int i=1;
            while(line!=null) {
                i++;
                line = in.readLine();
            }
        }
    }
}
```

Esempio

```
    video.println("Il file e' lungo " + (i-1) + "
    righe.");
}
catch(FileNotFoundException e) {
    video.println("Il file elenco.txt non esiste");
}
catch(IOException e) {
    video.println("Si e' verificata un'eccezione di IO");
}
}
}
```

Esempio

```
malchiod% java EccezioneGestita
Il file e' lungo 6 righe.
malchiod% rm elenco.txt
malchiod% java EccezioneGestita
Il file elenco.txt non esiste
malchiod%
```

throw VS catch

- Le parole chiave `throw` e `catch` sono complementari
 - un codice che utilizza `throw` gestisce le eccezioni che vengono lanciate all'interno dei suoi metodi **lanciandole** a sua volta al codice che lo ha chiamato
 - un codice che utilizza `catch` **cattura** le eccezioni che vengono lanciate all'interno dei suoi metodi

Eliminando un'eccezione

```
import java.io.*;
import prog.io.*;

class EccezioneMultipla {

    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();

        try {

            BufferedReader in =
                new BufferedReader(new FileReader("elenco.txt"));
            String line = in.readLine();
            int i=1;
            while(line!=null) {
                i++;
                line = in.readLine();
            }
            video.println("Il file e' lungo " + (i-1) + " righe.");
        }

        catch(IOException e) {
            video.println("Si e' verificata un'eccezione di IO");
        }
    }
}
```

Eliminando un'eccezione

```
malchiod% javac EccezioneMultipla.java
malchiod% java EccezioneMultipla
Si e' verificata un'eccezione di IO
malchiod%
```

- il programma viene compilato ugualmente...
...perché?

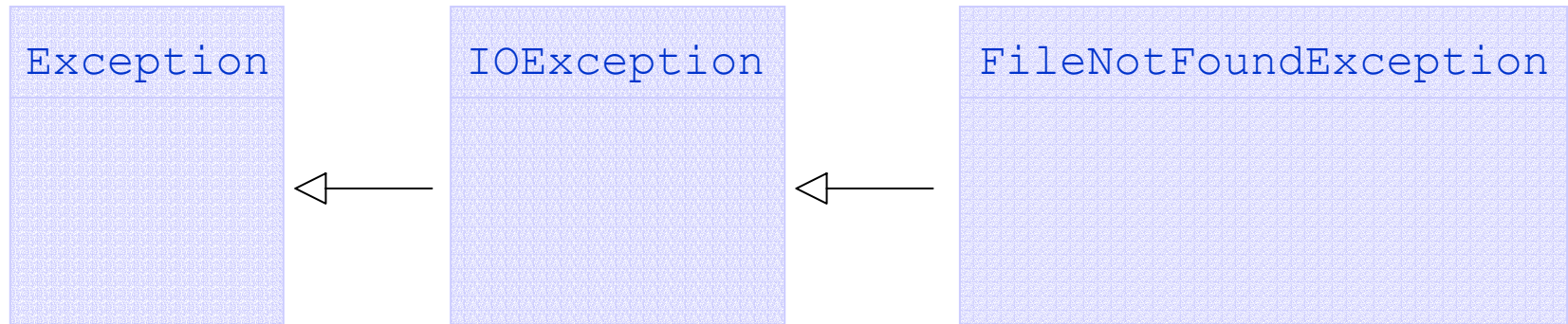
Gerarchia di eccezioni

- In Java le eccezioni che vengono lanciate quando si verifica un errore sono anche esse istanze di una classe.
- Tutte le classi che si riferiscono ad eccezioni sono sottoclassi della classe `Exception`
- Ereditare un'eccezione da un'altra eccezione permette di costruire una gerarchia di eccezioni

La classe `Exception`

- L'informazione più importante di una classe derivata da `Exception` è tipicamente contenuta nel suo nome
- La classe `Exception` e le sue derivate non implementano metodi particolari
 - `toString()` descrive l'eccezione
 - `printStackTrace()` stampa l'eccezione e la sequenza delle chiamate dei metodi che hanno portato a generarla

Esempio



- **Siccome** `FileNotFoundException` è interpretabile anche come `IOException`, la clausola `catch` del codice precedente la intercetta.

Gerarchia delle eccezioni

```
try {  
}  
catch(IOException e) {  
    video.println("Si e' verificata un'eccezione di IO");  
}  
catch(FileNotFoundException e) {  
    video.println("Il file elenco.txt non esiste");  
}
```

- `FileNotFoundException` verrebbe sempre gestita dalla prima clausola `catch` e la seconda sarebbe inutile
- Il compilatore riconosce questa situazione ed emette un errore

Esempio

```
import java.io.*;
import prog.io.*;

class EccezioneDoppia {
    public static void main(String[] args) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        try {
            BufferedReader in =
                new BufferedReader(new FileReader("elenco.txt"));
            String line = in.readLine();
            int i=1;
            while(line!=null) {
                i++;
                line = in.readLine();
            }
        }
    }
}
```

Esempio

```
    video.println("Il file e' lungo " + (i-1) + "
    righe.");
}
catch(IOException e) {
    video.println("Si e' verificata un'eccezione di IO");
}
catch(FileNotFoundException e) {
    video.println("Il file elenco.txt non esiste");
}
}
}
```

Esempio

```
malchiod% javac EccezioneDoppia.java
EccezioneDoppia.java:21: exception
    java.io.FileNotFoundException has already been caught
    catch(FileNotFoundException e) {
    ^
1 error
malchiod%
```

Costruire nuove eccezioni

- Estendendo la classe `Exception` o una delle classi derivate da essa è possibile costruire nuove eccezioni
- All'interno del codice è possibile lanciare eccezioni (esistenti o create appositamente) tramite il comando `throw`

MyException.java

```
class MyException extends Exception {  
    public MyException() {  
    }  
}
```

TestMyException.java

```
import prog.io.*;

class TestMyException {
    public static void main(String args[]) {
        ConsoleOutputManager video=new ConsoleOutputManager();
        try {
            MyException m = new MyException();
            throw m;
        } catch(MyException e) {
            video.println(e.toString());
            e.printStackTrace();
        }
    }
}
```

TestMyException

```
malchiod% java TestMyException
MyException
MyException
           at TestMyException.main(TestMyException.java:7)
malchiod%
```

TestStack.java

```
class TestStack {  
    public static void f() throws MyException {  
        throw new MyException();  
    }  
    public static void g() throws MyException {  
        f();  
        throw new MyException();  
    }  
}
```

TestStack.java

```
public static void main(String args[]) {  
    try {  
        f();  
    } catch (MyException e) {  
        e.printStackTrace();  
    }  
    try {  
        g();  
    } catch (MyException e) {  
        e.printStackTrace();  
    }  
}
```

TestStack

```
malchiod% java TestStack
MyException
    at TestStack.f(TestStack.java:3)
    at TestStack.main(TestStack.java:11)
MyException
    at TestStack.f(TestStack.java:3)
    at TestStack.g(TestStack.java:6)
    at TestStack.main(TestStack.java:16)
malchiod%
```

Runtime.java

```
class Runtime {  
    public static void main(String[] args) {  
        int i, j;  
        j=0;  
        i=3/j;  
    }  
}
```

Runtime.java

```
malchiod% javac Runtime.java
malchiod% java Runtime
Exception in thread "main" java.lang.ArithmeticException:
    / by zero
        at Runtime.main(Runtime.java:5)
malchiod%
```

- Perché la compilazione va a buon fine quando `main` non dichiara la possibilità di lanciare `ArithmeticException`?

RuntimeException

- **Eccezioni come**
`NullPointerException`,
`ArithmeticException` **e**
`ArrayIndexOutOfBoundsException`,
che riguardano tipicamente errori di
programmazione, sono raggruppate nella
superclasse `RuntimeException` **che**
non deve essere dichiarata dal
programmatore

Finally

```
class Switch {
    boolean state = false;
    void on() {state = true; }
    void off() {state = false; }
}

class OnOffException1 extends Exception {}
class OnOffException2 extends Exception {}
```

Finally

```
class TestSwitch {
    static Switch sw = new Switch();
    static void f() throws OnOffException1, OnOffException2 {}
    public static void main(String[] args) {
        try {
            sw.on();
            // Codice che può generare eccezioni...
            f();
            sw.off();
        } catch (OnOffException1 e) {
            sw.off(); // altro codice...
        } catch (OnOffException2 e) {
            sw.off(); // altro codice...
        }
    }
}
```

Se si verificano queste eccezioni l'interruttore viene spento

Cosa succede se si verifica un altro tipo di eccezione?

Finally

- In casi come questo risulta utile poter eseguire del codice indipendentemente dalla generazione di una qualsiasi eccezione nel blocco `try`
- Questo è possibile accodando ai vari blocchi `catch` un blocco `finally`

Finally

```
class TestSwitch {
    static Switch sw = new Switch();
    static void f() throws OnOffException1, OnOffException2 {}
    public static void main(String[] args) {
        try {
            sw.on();
            // Codice che può generare eccezioni...
            f();
        } catch (OnOffException1 e) {
            // altro codice...
        } catch (OnOffException2 e) {
            // altro codice...
        } finally {
            sw.off();
        }
    }
}
```

LetturaFile.java

```
import prog.io.*;
import java.io.*;

class LetturaFile {
    public static void main(String[] args) {
        ConsoleOutputManager out = new ConsoleOutputManager();
        int bytes = 0;
    }
}
```

LetturaFile.java

```
try {
    DataInputStream in = new DataInputStream(new
    FileInputStream("elenco.txt"));
    while(true) {
        in.readByte();
        bytes++;
    }
} catch(EOFException e) {
    out.println("Il file e' lungo " + bytes + " byte");
} catch(FileNotFoundException e) {
    out.println("Il file elenco.txt non esiste");
} catch(IOException e) {
    out.println("Si e' verificata un'eccezione di I/O");
}
}
```

LetturaDati.java

```
import prog.io.*;

class EndOfDataException extends Exception {
    public EndOfDataException() {
    }
}

class LetturaDati {
    public static void main(String[] args) {
        ConsoleOutputManager out = new ConsoleOutputManager();
        ConsoleInputManager in = new ConsoleInputManager();
        int somma = 0, dato;
```

LetturaDati.java

```
try {
    while(true) {
        out.println("Inserisci un intero (0 per uscire)");
        dato = in.readInt();
        if(dato != 0)
            somma += dato;
        else
            throw new EndOfDataException();
    }
} catch (EndOfDataException e) {
    out.println("Fine immisione dati");
    out.println("La somma dei numeri immessi e' " +
somma);
}
}
```