

Progettazione del Software

Giuseppe De Giacomo, Paolo Liberatore,
Massimo Mecella

Gestione delle eccezioni in Java



Condizioni di errore ...

Una condizione di errore in un programma può avere molte cause

- Errori di programmazione
 - Divisione per zero, cast non permesso, accesso oltre i limiti di un array
- Errori di sistema
 - Disco rotto, connessione remota chiusa, memoria non disponibile
- Errori di utilizzo
 - Input non corretti, tentativo di lavorare su file inesistente



... in Java

- Java ha una gerarchia di classi per rappresentare le varie tipologie di errore
 - dislocate in package diversi a seconda del tipo di errore
- La superclasse di tutti gli errori è la classe `Throwable` nel package `java.lang`
- Qualsiasi nuovo tipo di errore deve essere inserito nella discendenza di `Throwable`
 - solo su gli oggetti di questa classe si possono usare le parole chiave di Java per la gestione degli errori



3

Superclasse `Throwable`

- La superclasse `Throwable` ha due sottoclassi dirette, sempre in `java.lang`
 - `Error`
 - Errori fatali, dovuti a condizioni incontrollabili
 - Esaurimento delle risorse di sistema necessarie alla JVM, incompatibilità di versioni
 - In genere i programmi non gestiscono questi errori
 - `Exception`
 - Tutti gli errori che non rientrano in `Error`
 - I programmi possono gestire o no questi errori a seconda dei casi



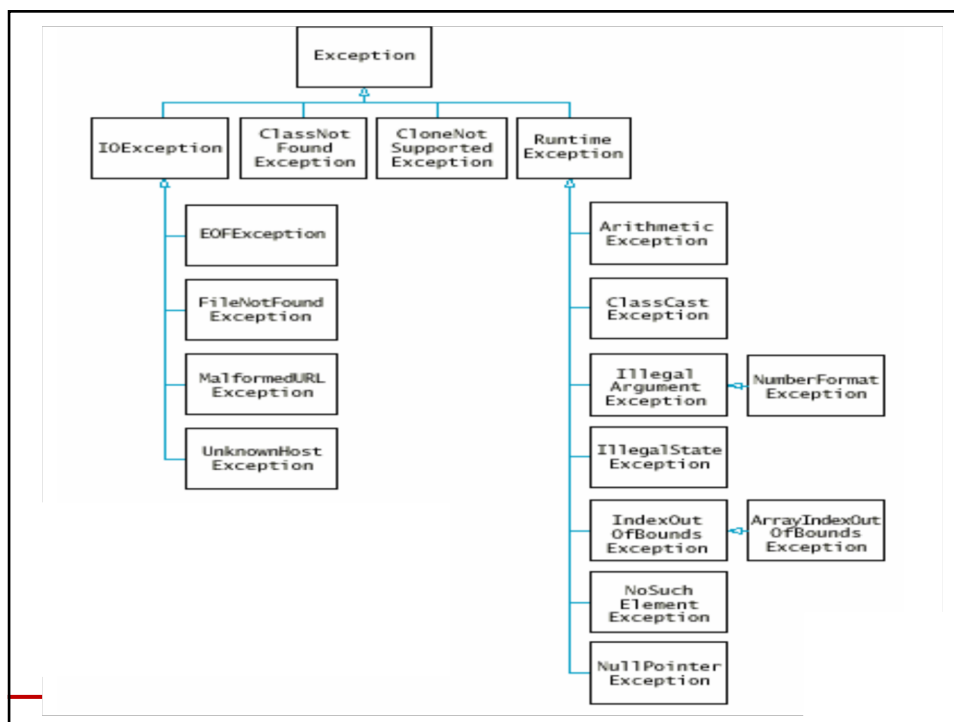
4

Eccezioni

- Una **eccezione** è un evento che interrompe la normale esecuzione del programma
- Se si verifica un'eccezione il metodo trasferisce il controllo ad un **gestore delle eccezioni**
 - Il suo compito è quello di uscire dal frammento di codice che ha generato l'eccezione e decidere cosa fare
- Java mette a disposizione varie classi per gestire le eccezioni, in vari package, ad es., `java.lang`, `java.io`
- Tutte le classi che gestiscono le eccezioni sono ereditate dalla classe `Exception`



5



Tipi di eccezioni (1)

- Due categorie
 - eccezioni controllate
 - dovute a circostanze esterne che il programmatore non può evitare
 - il compilatore vuole sapere cosa fare nel caso si verifichi l'eccezione
 - eccezioni non controllate
 - dovute a circostanze che il programmatore dovrebbe evitare scrivendo accuratamente il codice



7

Tipi di eccezioni (2)

- Esempio di eccezione controllata
 - `EOFException`: terminazione inaspettata del flusso di dati in ingresso
 - Può essere provocata da eventi esterni
 - errore del disco
 - interruzione del collegamento di rete
 - Il gestore dell'eccezione si occupa del problema



8

Tipi di eccezioni (3)

- Esempi di eccezione non controllata
 - `NullPointerException`: uso di un riferimento `null`
 - `IndexOutOfBoundsException`: accesso ad elementi esterni ai limiti di un array
- Non bisogna installare un gestore per questo tipo di eccezione
 - Il programmatore può prevenire queste anomalie, correggendo il codice



9

Le eccezioni non controllate indicano errori di logica causati dai programmatori e non da rischi esterni che non possono essere evitati. Per esempio, un'eccezione `NullPointerException` (eccezione di riferimento `null`) non è controllata. Praticamente qualsiasi metodo può sollevarne una; i programmatori non dovrebbero perder tempo a catturarle, ma fare in modo che nessun valore `null` venga dereferenziato.

A volte i programmatori devono usare giudizio per distinguere fra eccezioni controllate e non controllate. Si consideri per esempio la chiamata `Integer.parseInt(str)` che, se `str` non contiene un intero valido, solleva un'eccezione non controllata `NumberFormatException`. D'altro canto, `Class.forName(str)` solleva un'eccezione controllata `ClassNotFoundException` se `str` non contiene un nome di classe valido.

Perché questa differenza? Il motivo è che è possibile controllare se una stringa è un intero valido prima ancora di chiamare `Integer.parseInt`, mentre non è possibile sapere se una classe può essere caricata finché non ci si prova effettivamente.



10

Eccezioni controllate

- Tutte le sottoclassi di `IOException`
 - `EOFException`
 - `FileNotFoundException`
 - `MalformedURLException`
 - `UnknownHostException`
 - `ClassNotFoundException`
 - `CloneNotSupportedException`

11

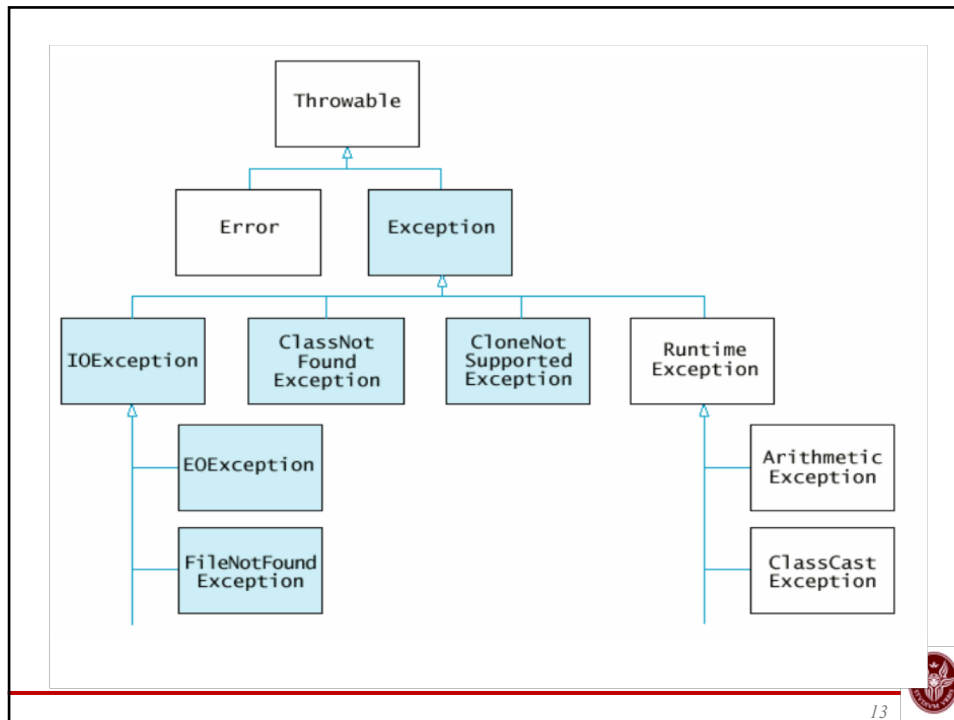


Eccezioni non controllate

- Tutte le sottoclassi di `RuntimeException`
 - `ArithmeticException`
 - `ClassCastException`
 - `IllegalArgumentException`
 - `IllegalStateException`
 - `IndexOutOfBoundsException`
 - `NoSuchElementException`
 - `NullPointerException`

12





Catturare eccezioni (1)

- Ogni eccezione deve essere gestita, altrimenti causa l'arresto del programma
- Per installare un gestore si usa l'enunciato `try`, seguito da tante clausole `catch` quante sono le eccezioni da gestire

```
try
{
    enunciato
    enunciato
    ...
}
catch (ClasseEccezione oggettoEccezione) {
    enunciato
    enunciato
    ...
}
catch (ClasseEccezione oggettoEccezione) {
    enunciato
    enunciato
    ...
}
...
```

Catturare eccezioni (2)

- Vengono eseguite le istruzioni all'interno del blocco `try`
- Se nessuna eccezione viene lanciata, le clausole `catch` sono ignorate
- Se viene lanciata una eccezione viene eseguita la corrispondente clausola `catch`

15



Dettagli su `Exception`

- Costruttori di oggetti della classe `Exception`:
 - `Exception()` costruisce un'eccezione senza uno specifico messaggio
 - `Exception(String msg)` costruisce un'eccezione con il messaggio `msg`
- Metodo ereditato dalla classe `Throwable`:
 - `String getMessage()` restituisce come stringa il messaggio contenuto nell'eccezione
- Quando l'eccezione viene lanciata:
 - può essere già stata creata o venir creata contestualmente
- Garbage collector: distrugge solo le eccezioni che sono state catturate

16




```

import java.io.*;
class ClasseEcc {
    void ff() throws IOException
    {
        Exception ec = new Exception("Prima eccezione");
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        int i = Integer.parseInt(br.readLine());
        try {
            if (i == 1) throw ec;
            throw new Exception("Altra eccezione");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        System.out.println("Prosecuzione della funzione");
    }

    public static void main (String[] args) throws IOException
    {
        ClasseEcc ex = new ClasseEcc();
        ex.ff();
        System.out.println("Prosecuzione del programma");
    }
}

```

17



```

$ java ClasseEcc
1
Prima eccezione
Prosecuzione della funzione
Prosecuzione del programma

$ java ClasseEcc
2
Altra eccezione
Prosecuzione della funzione
Prosecuzione del programma

```

18



Lanciare eccezioni

- Per lanciare un'eccezione, usiamo la parola chiave `throw` (*lancia*), seguita da un oggetto di tipo eccezione

```
throw exceptionObject;
```

- Il metodo termina immediatamente e passa il controllo al gestore delle eccezioni
 - Le istruzioni successive non vengono eseguite

19



Esempio

```
public class BankAccount
{
    public void withdraw(double amount)
    {
        if (amount > balance)
            throw new IllegalArgumentException("Saldo
            insufficiente");
        balance = balance - amount;
    }
    ...
}
```

La stringa in input al costruttore di `IllegalArgumentException` rappresenta il messaggio d'errore da associare all'eccezione

20



Segnalare eccezioni

- `BufferedReader.readLine()` può lanciare una `IOException`
- Un metodo che chiama `readLine()` può
 - gestire l'eccezione, cioè dire al compilatore cosa fare
 - non gestire l'eccezione, ma dichiarare di poterla lanciare
 - In tal caso, se l'eccezione viene lanciata, il programma termina visualizzando un messaggio di errore
- Per segnalare le eccezioni controllate che il metodo può lanciare usiamo la parola chiave `throws`

```
public void read(BufferedReader in) throws
    IOException
```

21



Esempio

```
public class Coin
{
    public void read(BufferedReader in) throws
        IOException
    {
        value = Double.parseDouble(in.readLine());
        name =in.readLine();
    }
    ...
}
```

La clausola `throws` segnala al chiamante di `Coin.read` che esso può generare un'eccezione di tipo `IOException`

22



Segnalare eccezioni

- Qualunque metodo che chiama `Coin.read` deve decidere se gestire l'eccezione o dichiarare di poterla lanciare

```
public class Purse
{
    public void read(BufferedReader in) throws
        IOException
    {
        while (...)
        { Coin c = new Coin();
          c.read(in);
          add(c);
        }
    }
    ...
}
```

23



Segnalare eccezioni

- Un metodo può lanciare più eccezioni controllate, di tipo diverso

```
public void read(BufferedReader in)
    throws IOException,
        ClassNotFoundException
```

24



Usare le eccezioni di Run Time

- Le eccezioni di runtime (`RuntimeException`) possono essere utilizzate per segnalare problemi dovuti ad input errati.
- Esempi:
 - Un metodo che preleva soldi da un conto corrente non può prelevare una quantità maggiore del saldo
 - Un metodo che effettua una divisione non può dividere un numero per zero

25



Progettare nuove eccezioni (1)

- Se nessuna delle eccezioni di runtime ci sembra adeguata al nostro caso, possiamo progettarne una nuova.
- I nuovi tipi di eccezioni devono essere inseriti nella discendenza di `Throwable`, e in genere sono sottoclassi di `RuntimeException`
- Un tipo di eccezione che sia sottoclasse di `RuntimeException` sarà a controllo non obbligatorio

26



Progettare nuove eccezioni (2)

- Introduciamo un nuovo tipo di eccezione per controllare che il denominatore sia diverso da zero, prima di eseguire una divisione:

```
public class DivisionePerZeroException extends
RuntimeException{
    public DivisionePerZeroException() {
        super("Divisione per zero!");
    }
    public DivisionePerZeroException(String msg)
    { super(msg); }
}
```

27



...

```
public class Divisione {
    public Divisione(int n, int d) {
        num=n;
        den=d;
    }
    public double dividi(){
        if (den==0)
            throw new DivisionePerZeroException();
        return num/den;
    }
    private int num;
    private int den;
}
```

28



.....

```
public class Test {  
    public static void main(String[] args) throws  
        IOException  
    {  
        double res;  
        Scanner scan = new Scanner(System.in);  
        System.out.print("Inserisci il numeratore:");  
        int n= Integer.parseInt(scan.nextLine());  
        System.out.print("Inserisci il denominatore:");  
        int d= Integer.parseInt(scan.readLine());  
        Divisione div = new Divisione(n,d);  
        res = div.dividi();  
    }  
}
```

29



.....

```
Inserisci il numeratore: 5  
Inserisci il denominatore: 0  
DivisionePerZeroException: Divisione per zero!  
at Divisione.dividi(Divisione.java:12)  
at Test.main(Test.java:22)  
Exception in thread "main"
```

- Il main invoca il metodo `dividi` della classe `Divisione` alla linea 22
- Il metodo `dividi` genera una eccezione alla linea 12

30



.....

```
public class Test {  
    public static void main(String[] args) throws  
        IOException {  
        double res;  
        Scanner scan = new Scanner(System.in);  
        System.out.print("Inserisci il numeratore:");  
        int n= Integer.parseInt(scan.nextLine());  
        System.out.print("Inserisci il denominatore:");  
        int d= Integer.parseInt(scan.readLine());
```

31



.....

```
try  
{  
    Divisione div = new Divisione(n,d);  
    res = div.dividi();  
    System.out.print(res);  
} catch (DivisionePerZeroException exception)  
{  
    System.out.println(exception);  
}  
}
```

32



.....

- Cosa fa l'istruzione

`System.out.println(exception); ?`

- Invoca il metodo `toString()` della classe

`DivisionePerZeroException`

- Ereditato dalla classe `RuntimeException`

- Restituisce una stringa che descrive l'oggetto

Exception costituita da

- Il nome della classe a cui l'oggetto appartiene
seguito da `:` e dal messaggio di errore associato
all'oggetto

33



.....

Inserisci il numeratore:5

Inserisci il denominatore:0

`DivisionePerZeroException: Divisione per zero!`

- `DivisionePerZeroException` è la classe a cui l'oggetto `exception` appartiene

- `Divisione per zero!` è il messaggio di errore associato all'oggetto `exception` (dal costruttore)

34



Catturare eccezioni

- Per avere un messaggio di errore che stampa lo stack delle chiamate ai metodi in cui si è verificata l'eccezione usiamo il metodo `printStackTrace()`

```
catch (DivisionePerZeroException exception)
{
    exception.printStackTrace();
}
```

35



La clausola `finally` (*finally* 😊)

- Il lancio di un'eccezione arresta il metodo corrente
- A volte vogliamo eseguire altre istruzioni prima dell'arresto
- La clausola `finally` viene usata per indicare un'istruzione/blocco che va eseguita/o sempre

```
try {
    enunciato
    enunciato
    ...
} finally {
    enunciato
    enunciato
    ...
}
```

36

