

# 18 - Eccezioni

Programmazione e analisi di dati  
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa  
<http://pages.di.unipi.it/milazzo>  
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica  
A.A. 2017/2018

# Situazioni anomale a run-time (1)

Java è un linguaggio **fortemente tipato**

- 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 (indice negativo o maggiore della dimensione)
- Errori aritmetici (esempio: divisione per zero)
- Errori di formato: si chiede all'utente un intero e l'utente inserisce una stringa

## Situazioni anomale a run-time (2)

Un po' di esempi:

```
public class ErroreArray {  
    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");  
    }  
}
```

## Situazioni anomale a run-time (3)

```
import java.util.Scanner;

public class ErroreAritmetico {

    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??
    }
}
```

## Situazioni anomale a run-time (4)

```
import java.util.Scanner;

public class ErroreFormato {

    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);

    }
}
```

# 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
  - 3a. passa il controllo a un **gestore** di eccezioni (implementato dal programmatore)
  - 3b. se il programmatore non ha previsto nessun gestore, interrompe il programma e stampa il messaggio di errore

Bene... come si fa quindi a implementare un gestore di eccezioni?

- Tramite il costrutto `try-catch`

## 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) nella porzione di programma monitorata (gestione dell'eccezione)

Si usa così:

```
// ... comandi non monitorati ....  
  
try {  
    // ... comandi monitorati ....  
}  
catch (Exception e) {  
    // ... comandi da eseguire in caso di eccezione  
}  
  
// ... altri comandi non monitorati ....
```





# Gestire eccezioni (1)

Aggiungiamo un gestore delle eccezioni alla classe ErroreAritmetico

```
import java.util.Scanner;

public class ErroreAritmetico2 {

    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");
    }
}
```

## Gestire eccezioni (2)

Altro esempio: la classe ErroreFormato...

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

public class ErroreFormato2 {
    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);
    }
}
```

## Gestire eccezioni (3)

Un costrutto try-catch può gestire più tipi di eccezione contemporaneamente

```
try {  
    ....  
}  
catch (NumberFormatException e) {  
    ....  
}  
catch (Exception e) {  
    ....  
}
```

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... (vedere documentazione)

## Gestire eccezioni (4)

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
  - ▶ ad esempio: 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!

- esempio: la classe `ErroreArray` lanciava un'eccezione a causa di un errore nel ciclo `for`

## Eccezioni checked e unchecked

In alcuni casi, inoltre, il compilatore **obbliga** a definire un gestore di eccezioni.

Le eccezioni si dividono in:

- **Checked** (o **controllate**) per le quali il compilatore richiede che ci sia un gestore
- **Unchecked** (o **non controllate**) per le quali il gestore non è obbligatorio

Tutte le eccezioni che abbiamo visto fino ad ora sono unchecked!

Per essere unchecked un'eccezione deve essere una sottoclasse di `RuntimeException`, altrimenti è checked

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 (vedremo)

# Lanciare eccezioni (1)

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 definita dal programmatore stesso
- `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 ArithmeticException();
```

```
throw new EccezionePersonalizzata();
```

## Lanciare eccezioni (2)

Il comando `throw` si può usare direttamente dentro un `try-catch`

```
try {  
    .....  
    throw new Exception("errore generico");  
    .....  
}  
catch(Exception e) {  
    ...  
}
```

Ma in realtà l'uso più sensato di `throw` è all'interno dei metodi...

## Lanciare eccezioni (3)

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



## Lanciare eccezioni (4)

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**

```
public void preleva(int somma)
    throws IOException, IllegalArgumentException { ... }
```

- **Attenzione** alla `s` finale:
  - ▶ `throws` si usa nell'intestazione del metodo
  - ▶ `throw` si usa all'interno (nel punto in cui si verifica l'errore)

## Lanciare eccezioni (4)

Esempio: controllo correttezza parametri

```
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;  
    }  
  
}
```

## Lanciare eccezioni (5)

Dove la classe `EccezioneBaseNegativa` è definita banalmente così:

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