Eco-design Digitale di Base per i servizi ICT

Programmazione in Java

Massimo Giaccone, Luglio 2025

Obiettivi della lezione

- Comprendere il funzionamento delle eccezioni
- Gestire errori con try-catch-finally
- Distinguere tra eccezioni checked e unchecked
- Leggere e scrivere su file con le principali classi Java

Cos’è un eccezione

- Un'**eccezione** rappresenta un evento anomalo che si verifica durante l'esecuzione del programma e che interrompe il flusso normale delle istruzioni.

- Java fornisce un sistema robusto per la gestione degli errori tramite il meccanismo delle eccezioni.

- Esempio:
  - **int a = 5 / 0;** // ArithmeticException

Try-catch: struttura base

- La struttura try-catch consente di intercettare ed eventualmente gestire un errore che si potrebbe verificare all'interno del blocco try.

```
class Try {

    Run main | Debug main
    public static void main(String[] args) {

        try {
            //codice potenzialmente errato
        } catch (Exception e) {
            //codice di gestione dell'errore
        }
    }
}
```

Esercizio 1

Scrivi un programma che accetti due numeri interi come parametri (utilizzare gli argomenti del programma) e ne stampi il quoziente. Utilizza try-catch per evitare il crash in caso di divisione per zero.

Blocco finally

- Il blocco finally, se presente, viene sempre eseguito dopo l'esecuzione del try e/o catch, **indipendentemente dal fatto che si sia verificata o meno un'eccezione**.
- Utile per rilasciare risorse: chiudere file, connessioni.

```java
class Try {

    Run main | Debug main
    public static void main(String[] args) {

        try {
            //codice potenzialmente errato
        } catch (Exception e) {
            //codice di gestione dell'errore
        } finally {
            // codice sempre eseguito
        }
    }
}
```

Checked vs Unchecked

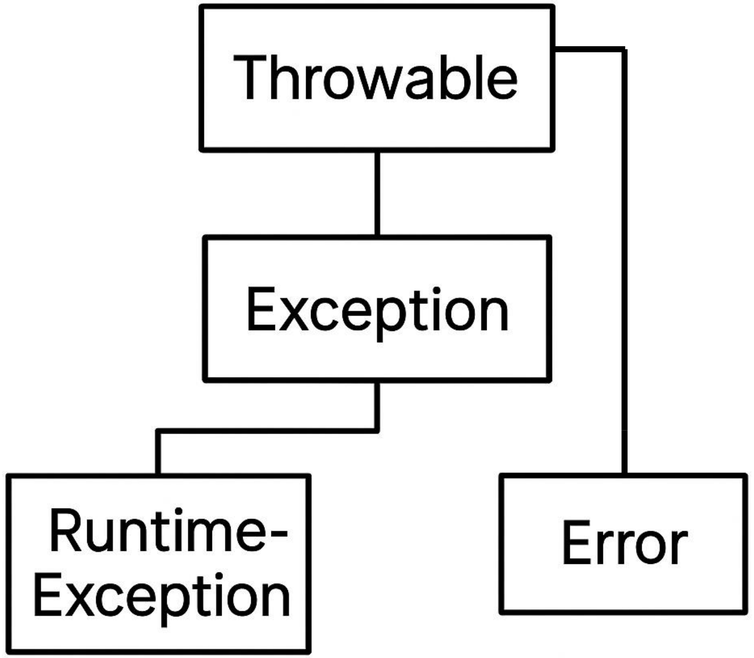
Java distingue tra **eccezioni verificate (checked)** e **non verificate (unchecked)**:

- **Checked:** devono essere dichiarate o catturate (es. IOException)
- **Unchecked:** derivate da RuntimeException, non obbligano al try-catch

Esempi:

- IOException, SQLException → checked
- NullPointerException, ArrayIndexOutOfBoundsException → unchecked

Gerarchia eccezioni Java



- Solo Exception è progettata per essere catturata con try-catch
- Error rappresenta problemi di sistema gravi (es. OutOfMemoryError)

Esercizio 2

Scrivi un metodo che accetta una stringa come parametro e stampa la sua lunghezza. Se la stringa è null, cattura la NullPointerException e stampa un messaggio di avviso.

Creazione di eccezioni personalizzate

In Java possiamo creare le nostre eccezioni personalizzate estendendo la classe Exception o RuntimeException.

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

Uso di eccezioni personalizzate

```
int x = Integer.parseInt(args[0]);

 try {
     if (x < 0) {
         throw new NumeroNegativoException("Il numero è negativo!");
     }
     System.out.println(x);
 } catch (NumeroNegativoException e) {
     System.out.println("Eccezione: " + e.getMessage());
 }
```

Esercizio 3

Crea una classe Salvadanaio che contiene un saldo. Scrivi un metodo preleva(int x) che lancia un'eccezione SaldoInsufficienteException se x > saldo.

Intro all’I/O in Java

Java fornisce classi per leggere e scrivere file in modo efficiente e sicuro.

Le più usate sono:

- Lettura: Scanner, BufferedReader
- Scrittura: FileWriter, BufferedWriter

Tutte le operazioni I/O sono potenzialmente soggette a errori → IOException

Lettura da File con Scanner

```
Scanner scanner = new Scanner(new File("dati.txt"));
    while (scanner.hasNextLine()) {
        String riga = scanner.nextLine();
        System.out.println(riga);
    }
    scanner.close();
```

- Scanner è semplice e comodo, ma non ottimale per file molto grandi

Esercizio 4

Scrivi un programma che legge da un file chiamato nomi.txt e stampa ogni riga sulla console.

Leggere da file con Buffered Reader

```
BufferedReader br = new BufferedReader(new FileReader("file.txt"));
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
br.close();
```

- Più efficiente per file di grandi dimensioni rispetto a Scanner

Scrittura file con FileWriter

```
FileWriter fw = new FileWriter("output.txt");
fw.write("Ciao mondo\n");
fw.close();
```

- Se il file esiste, viene sovrascritto

Scrittura file con Buffered Writer

```
BufferedWriter bw = new BufferedWriter(new FileWriter("out.txt"));
bw.write("Hello!\n");
bw.newLine();
bw.close();
```

- Consigliato per scrittura su file con più righe

Esercizio 5

Scrivi su un file risultato.txt le righe:

Riga 1

Riga 2 ...

Riga 10

Esercizio 6

Scrivi un programma che legge da input.txt, converte ogni riga in MAIUSCOLO e la salva in output.txt

Gestione eccezioni I/O in Java

- Le classi di I/O lanciano IOException se il file non esiste, è bloccato o c'è un errore di accesso

- Gestire con try-catch oppure dichiarare throws IOException

Esercizio 7

Scrivi un programma che: 1. Legge numeri interi da numeri.txt 2. Calcola e stampa la loro somma 3. Gestisce errori di file mancante e numeri mal formattati