Exceptions & I/O in Java

Exceptions

L'eccezione permette di segnalare casi di errore e conseguentemente gestirli, senza il ritorno di un valore in una situazione normale. Permette una maggiore affidabilità in fase di testing per errori non espliciti.

In Java le eccezioni sono OGGETTI di tipo java.lang.Throwable e solo oggetti di sottoclassi sue possono essere lanciati.

Sintassi: throw e , con e espressione di tipo T ≤ Throwable per lanciare l'eccezione, il flusso normale delle operazioni viene interrotto

Solitamente "throw new object", creo un nuovo oggetto.

oppure

un metodo ausiliario che lancia eccezione: IllegalArgumentException()

Una volta lanciata l'eccezione si passa a un frammento di codice che viene eseguito in tali casi eccezionali

→ se non gestisco l'eccezione, il programma si interrompe comunque mostrando la sorgente dell'errore.

Eseguire un blocco dove qualcosa può andare storto e catturare l'eventuale eccezione si riassume con il blocco

try - catch

Example

```
public class ExceptionTest {
    public static void main(String[] args) {
        var timer = new TimerClass(30);
        try {
            timer.reset(Integer.parseInt(args[0]));
        } catch (Throwable e) {
            timer.reset(0);
        }
        System.out.println(timer.getTime()); // the statement is always executed
    }
}
```

0 < x < 60

non numero ✓
no args

eccezione generica

Gestire l'eccezione (con il catch) evita il crash del programma (e rispetto a un return evita che il programma si concluda)

* Posso catturare eccezioni di vario genere utilizzando più catch (il meno specifico in fondo, ovvero Throwable e, che cattura qualsiasi altra eccezione non prevista manualmente)

Si cattura solo in un catch, se nessuna clausola si verifica propago l'eccezione come se il try-catch non esistesse (torno su della funzione finché non viene catturato / terminato il programma)

Se Throwable non è in fondo il compilatore lo segnala.

Le eccezioni si dividono in unchecked e checked

- unchecked → errori 'fisici' (out of memory)
- unchecked → errori logici
- checked → controllate dal compilatore (utilizzando try-catch ad esempio)

Si può forzare il compilatore a catturare le eccezioni checked, nel caso in cui si sa che si può mancare una clausola di catch.

Questo è possibile tramite throws dopo l'intestazione della funzione + il tipo di eccezione. Posso non catturare qualcosa che quindi deve essere

catturato a livello superiore.

Example 2: error better handled at an higher level

```
static void read_throws(BufferedReader br) throws IOException {
    String line;
    do {
        line = br.readLine(); // may throw IOException, 'throws' clause needed!
        if (line != null)
            System.out.println(line);
    } while (line != null);
}
public void caller(BufferedReader br) {
    try { // the caller has more control on method 'read'
        read_throws(br);
    } catch (IOException e) {
        System.err.println(e.getMessage());
        ... // asks the user another file to read
    }
    ...
}
```

Input/output contenuto java.io

Si hanno

→ stream in/out binari : InputStream, OutputStream

→ stream in/out di caratteri : Reader, Writer

Molte classi implementano il decorator design pattern: modo dinamico per estendere le caratteristiche di un oggetto in modo dinamico, su singoli oggetti, senza inheritance.

L'oggetto viene "avvolto" dentro un altro, potendo usare caratteristiche di entrambe. Useremo Reader e PrintWriter con funzionalità di conversioni già implementate.

- readLine legge linee di caratteri
- println stampa linee di caratteri

Quando un file viene aperto, a fine elaborazione su esso, va anche chiuso.

Si può fare con finally, in questo anche la chiusura può lanciare eccezione. È un blocco che viene posto dopo try-catch e viene eseguito sempre. → es. finally { try close (file); }

Nelle ultime versioni di Java viene omesso: vengono rilasciate automaticamente le risorse quando non più necessarie

→ try with resources (utilizzato quando si lavora con file)

Example with try-with-resources

```
public static void main(String[] args) {
   try (var br = tryOpen(args)) { // br has type BufferedReader ≤ AutoCloseable
      read(br);                   // may throw IOException
   } catch (IOException e) {
      System.err.println(e.getMessage());
   }
}
```

try close() non necessario

eventuali errori fra (...) vengono catturati comunque dai catch