

GESTIONE DEGLI ERRORI

- Spesso vi sono **istruzioni “critiche”**, che in certi casi possono produrre errori.
- L’approccio classico consiste ***nell’inserire controlli*** (if... else..) per ***cercare di intercettare a priori*** le situazioni critiche
- Ma è un modo di procedere spesso ***insoddisfacente***
 - non è facile prevedere tutte le situazioni che potrebbero produrre l’errore
 - “gestire” l’errore spesso significa solo stampare a video un messaggio.

ECCEZIONI

Java introduce il concetto di **eccezione**

- anziché *tentare di prevedere* le situazioni di errore, *si tenta di eseguire l'operazione in un blocco controllato*
- se si produce un errore, l'operazione ***solleva un'eccezione***
- l'eccezione viene *catturata* dal blocco entro cui l'operazione è eseguita...
- ... e può essere *gestita* nel modo più appropriato.

ECCEZIONI

```
try {  
    /* operazione critica che può  
       sollevare eccezioni */  
}  
  
catch (Exception e) {  
    /* gestione dell'eccezione */  
}
```

- Se l'operazione solleva *diversi tipi* di eccezione in risposta a diversi tipi di errore, *più blocchi catch* possono seguire lo stesso blocco **try**

ESEMPIO

Conversione stringa / numero

- In Java, la conversione stringa / numero intero è svolta dal metodo statico

```
int Integer.parseInt(String s)
```

- L'operazione è critica, perché può avvenire *solo se la stringa data contiene la rappresentazione di un intero*
- Se ciò non accade, `parseInt` **solleva una `NumberFormatException`**

ESEMPIO

```
class EsempioEccezione {  
    public static void main(String args[]) {  
        int a = 0;  
        String s = "1123";  
  
        try {  
            a = Integer.parseInt(s);  
        }  
        catch (NumberFormatException e) {  
            System.out.println("Stringa mal fatta");  
        }  
    }  
}
```

ESEMPIO

```
class EsempioEccezione {  
    public static void main(String args[]) {  
        int a = 0;  
        String  
        try {  
            a =  
        }  
        catch (NumberFormatException e) {  
            System.out.println("Stringa mal fatta");  
        }  
    }  
}
```

Catturare le eccezioni è spesso importante:
un'eccezione non catturata *si propaga verso l'esterno*, di blocco in blocco: se raggiunge il **main**, provoca l'aborto del programma

COS'È UNA ECCEZIONE

- Una eccezione è *un oggetto*, istanza di `java.lang.Throwable` o di una sua sottoclasse.
- Le due sottoclassi più comuni sono `java.lang.Exception` e `java.lang.Error`
- La parola “eccezione” è però spesso riferita a entrambe

COS'È UNA ECCEZIONE

- Un **Error** indica problemi relativi al caricamento della classe o al funzionamento della macchina virtuale Java (es. not enough memory), e va considerato *irrecuperabile*: perciò *non* è da catturare.
- Una **Exception** indica invece situazioni *recuperabili*, almeno in linea di principio (fine file, indice di un array oltre i limiti, errori di input, etc.): *va catturata e gestita*.

ECCEZIONI COME OGGETTI

- Poiché un'eccezione è un oggetto, può *contenere dati o definire metodi*
- Tutte le eccezioni definiscono un metodo `getMessage()` che restituisce il messaggio d'errore associato
- Alcune eccezioni definiscono dei campi dati (ad esempio, `bytesTransferred` in `InterruptedException`) che danno altre informazioni, utili per gestire la situazione.

RILANCIO DI ECCEZIONI

In Java, un metodo che possa generare un'eccezione deve:

- *o gestire l'eccezione*, con un costrutto `try / catch`
- *oppure rilanciarla esplicitamente all'esterno del metodo*, delegandone in pratica la gestione ad altri.

Se sceglie questa seconda strada, il metodo *deve* indicare quale eccezione può “uscire” da esso, con la clausola **throws**

RILANCIO DI ECCEZIONI

Ad esempio, un metodo che svolga una conversione stringa/numero, **anziché gestire la `NumberFormatException`** può decidere di ***rilanciarla all'esterno***:

```
public int readInteger(String s)
    throws NumberFormatException {
    return Integer.parseInt(s) ;
}
```

Può sollevare un'eccezione

Non la gestisce, la rilancia all'esterno

DEFINIRE NUOVE ECCEZIONI

Dato che un'eccezione è un normale oggetto Java, è possibile:

- **definire nuovi tipi di eccezione** definendo *nuove classi*
- **generare eccezioni** dall'interno di propri metodi.

Per definire un nuovo tipo di eccezione basta **definire una nuova classe** che **estenda la classe base Exception** (o una delle sue sottoclassi)

ESEMPIO

```
class NumberTooBigException
    extends IllegalArgumentException {
    public NumberTooBigException() {
        super();
    }
    public NumberTooBigException(String s) {
        super(s);
    }
}
```

Definisce i due costruttori standard:

- quello di default
- quello con un parametro stringa (il messaggio)

ESEMPIO

Per sollevare (generare) un'eccezione:

- prima si crea l'oggetto eccezione da lanciare
- poi lo si lancia con l'istruzione **throw**

```
public int readInteger(String s)
    throws NumberFormatException,
           NumberTooBigException {
    int x = Integer.parseInt(s);
    if (x>100)
        throw new NumberTooBigException();
    return x;
}
```

ESEMPIO

Per sollevare un'eccezione:

- **prim**
- **poi**

Importante non confondere:

- la **clausola throws** che dichiara che un metodo rilancia all'esterno un'eccezione
- l'**istruzione throw** che solleva un'eccezione

```
public int Integer.parseInt(String s)
    throws NumberFormatException,
           NumberTooBigException {
    int x = Integer.parseInt(s);
    if (x>100)
        throw new NumberTooBigException();
    return x;
}
```