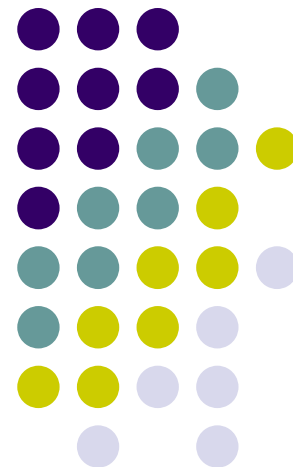
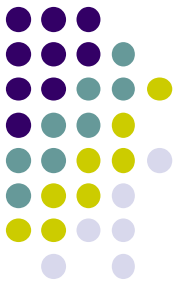


Corso di Programmazione

Gestione delle eccezioni *Concetti base*

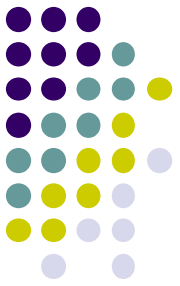


Il problema



- In alcuni casi, chi scrive una libreria può rilevare errori a run-time, ma non sa come gestirli
- Chi usa la libreria, invece, sa come gestire tali errori ma non può rilevarli
- Soluzioni ad-hoc possono essere definite di volta in volta
- È necessario un meccanismo generale, semplice da usare ed elegante per gestire situazioni “eccezionali”

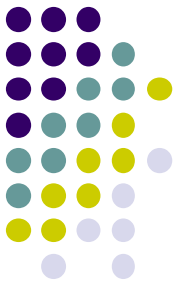
Esempio: overflow a seguito di somma



```
public static void main(String[] args) {  
    int x = Integer.MAX_VALUE;  
    int y = 34;  
    System.out.println(somma(x,y));  
}
```

```
public static int somma(int x, int y){  
    if ((x>0 && y>0 && x > Integer.MAX_VALUE-y) ||  
        (x<0 && y<0 && x < Integer.MIN_VALUE-y))  
        // che facciamo???  
    return x+y;  
}
```

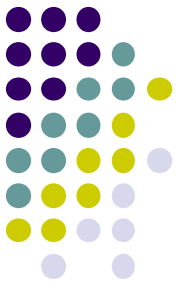
Soluzione 1: termina il programma



```
public static int somma(int x, int y){  
    if ((x>0 && y>0 && x > Integer.MAX_VALUE-y) ||  
        (x<0 && y<0 && x < Integer.MIN_VALUE-y))  
        System.exit(0); //termina il programma  
    return x+y;  
}
```

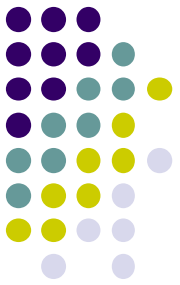
Vogliamo gestire il problema, non terminare il programma, Spesso è desiderabile che il programma possa continuare a funzionare, per poter almeno salvare i dati elaborati fino a quel momento, o rilasciare delle risorse

Soluzione 2: restituisce un valore di errore

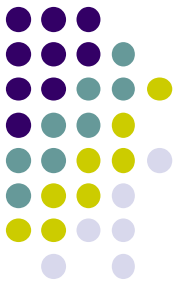


- Non sempre possibile
 - Nessun valore del tipo `int` può essere usato per indicare una situazione eccezionale
- Si può utilizzare una variabile booleana passata per riferimento
 - Diventa complicato quando possono presentarsi diverse situazioni eccezionali
 - Inoltre ciò altererebbe la semantica da “operatore” della funzione, perchè la somma binaria richiede esattamente due operandi

Il meccanismo di gestione delle eccezioni



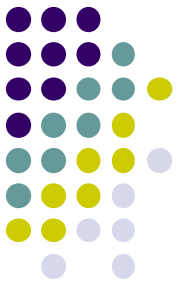
- Fornisce un'alternativa alle tecniche tradizionali quando queste sono insufficienti e prone ad errori
- L'idea è che una funzione che trova un errore che non sa gestire *lancia* (throw) un'eccezione, nella speranza che il suo chiamante (diretto o indiretto) possa gestire il problema
- Consente di separare il codice per la gestione dell'eccezione dal codice “ordinario”
 - Il programma diventa più leggibile



Gestione delle eccezioni

- Una funzione che vuole gestire un certo tipo di eccezione (handler) può farlo indicando che intende *catturare* (catch) quel tipo di eccezione
- Se viene lanciata un'eccezione e ripercorrendo a ritroso la catena di chiamanti non si incontra nessuna funzione che cattura l'eccezione, il programma viene terminato

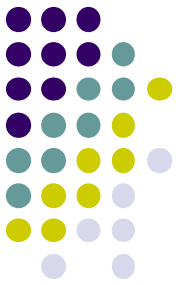
Gestione delle eccezioni



```
public static int somma(int x, int y){  
    if ((x>0 && y>0 && x >  
        Integer.MAX_VALUE-y) ||  
        (x<0 && y<0 && x < Integer.MIN_VALUE-  
        y))  
        //System.exit(0);  
        throw new  
        IllegalArgumentException("Overflow!");  
    return x+y;  
}
```

- La funzione somma *lancia* un'eccezione al verificarsi di una situazione che non sa gestire
 - throw [*oggetto*]
- Ovviamente, **throw fa terminare l'esecuzione della funzione somma**

Gestione delle eccezioni

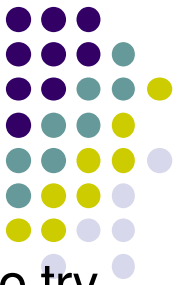


```
public static void main(String[] args) {  
    int x = Integer.MAX_VALUE;  
    int y = 34;  
    try {  
        System.out.println(somma(x,y));  
    }  
    catch (IllegalArgumentException e) {  
        System.out.println("Overflow!");  
    }  
    System.out.println("non sono terminato!");  
}
```

Il programma gestisce il problema e può eventualmente continuare la sua esecuzione

- Il costrutto **catch()** (*exception handler*) può essere usato solo dopo un blocco preceduto dalla keyword **try** o dopo un altro blocco **catch()**
- **catch()** ha tra parentesi una dichiarazione *simile* a quella degli argomenti di una funzione

Gestione delle eccezioni



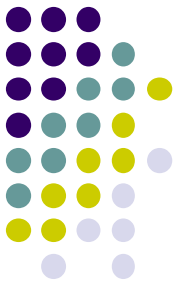
```
public static void main(String[] args) {  
    int x = Integer.MAX_VALUE;  
    int y = 34;  
    try {  
        System.out.println(somma(x,y));  
    }  
    catch (IllegalArgumentException e) {  
        System.out.println("Overflow!");  
    }  
    System.out.println("non sono  
terminato!");  
}
```

- Se una funzione nel blocco try lancia un'eccezione
 - Le istruzioni nel blocco try seguenti tale funzione non vengono eseguite
 - Viene eseguito (se esiste) solo il primo handler trovato per il tipo di eccezione lanciata



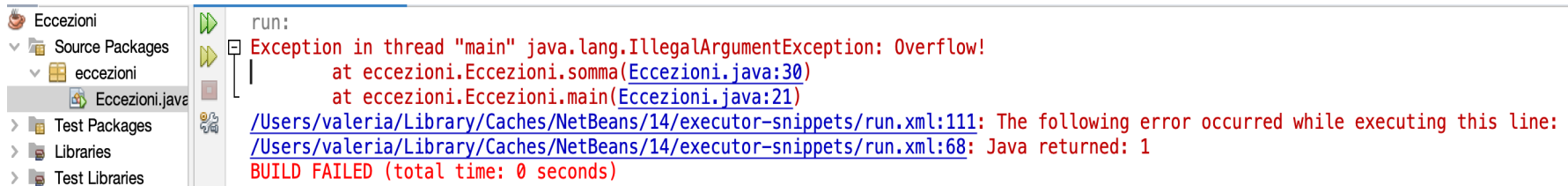
```
run:  
Overflow!  
non sono terminato!  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Cosa succede se non c'è il try-catch?

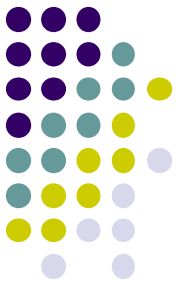


```
public static void main(String[] args) {  
    int x = Integer.MAX_VALUE;  
    int y = 34;  
    System.out.println(somma(x,y)); //line 21  
}
```

```
public static int somma(int x, int y){  
    if ((x>0 && y>0 && x > Integer.MAX_VALUE-y) ||  
        (x<0 && y<0 && x < Integer.MIN_VALUE-y))  
        throw new IllegalArgumentException("Overflow!"); //line  
    30  
    return x+y;  
}
```

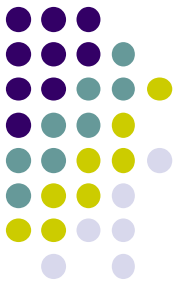


Stack Trace



- Le informazioni che vengono mostrate a seguito dell'esecuzione sono dette traccia dello stack (stack trace)
- Includono:
 - Il nome dell'eccezione (IllegalArgumentException)
 - Un messaggio descrittivo del problema
 - La catena completa delle chiamate, cioè il percorso di esecuzione che ha portato al verificarsi del problema
 - Partendo dall'ultima riga della traccia dello stack si evince che l'eccezione è stata rilevata alla riga 21 del metodo main
 - Ogni riga della traccia dello stack contiene il nome qualificato della classe e il metodo seguito dal nome del file e dal numero di riga
 - La riga più in alto della catena indica il **throw point**

Esempio: divisione per zero



```
public static void main(String[] args) {  
    Scanner input = new Scanner(System.in);  
    System.out.println("Inserire due numeri interi: ");  
    int x = input.nextInt();  
    int y = input.nextInt();  
    System.out.println("Risultato della divisione intera " + x + "/" + y);  
    System.out.println(DivisionePerZero(x,y));  
}
```

```
public static int DivisionePerZero(int a, int b){  
    return a/b;  
}
```

Test...




```
run:
Inserire due numeri interi:
3
0
Risultato della divisione intera 3/0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at eccezioni.Eccezioni2.DivisionePerZero(Eccezioni2.java:28)
    at eccezioni.Eccezioni2.main(Eccezioni2.java:24)
/Users/valeria/Library/Caches/NetBeans/14/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/valeria/Library/Caches/NetBeans/14/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 5 seconds)
```

```
run:
Inserire due numeri interi:
4
r
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at eccezioni.Eccezioni2.main(Eccezioni2.java:22)
/Users/valeria/Library/Caches/NetBeans/14/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/valeria/Library/Caches/NetBeans/14/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 15 seconds)
```

Osservazioni



- In Java la divisione per zero tra valori interi genera automaticamente una eccezione di tipo *ArithmeticException*
 - Quindi l'esecuzione del metodo *DivisionePerZero* può generare una eccezione anche se non abbiamo esplicitamente introdotto una *throw* nel suo codice!
- Quando il metodo *nextInt* di *Scanner* riceve una stringa che non rappresenta un intero valido solleva una eccezione di tipo *InputMismatchException*
- Il programma *main* deve gestire **due diversi tipi di eccezione!**



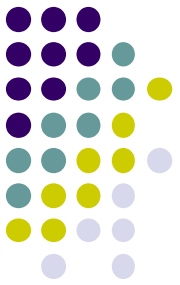
```
import java.util.Scanner;
import java.util.InputMismatchException;

public class Eccezioni2 {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.println("Inserire due numeri interi: ");
        try {
            int x = input.nextInt();
            int y = input.nextInt();
            int ris = DivisionePerZero(x,y);
            System.out.println("Risultato della divisione intera " + x + "/" + y + ":" + ris);
        }
        catch(InputMismatchException e){
            System.err.printf("Eccezione: %s%n", e);
            System.err.println("Non hai inserito due INTERI...");
        }
        catch(ArithmeticException e){
            System.err.printf("Eccezione: %s%n", e);
            System.err.println("Divisione per zero!");
        }
    }
}
```

Test...



run:

Inserire due numeri interi:

4

0

Eccezione: java.lang.ArithmeticException: / by zero
Divisione per zero!

BUILD SUCCESSFUL (total time: 5 seconds)



run:

Inserire due numeri interi:

4

w

Eccezione: java.util.InputMismatchException
Non hai inserito due INTERI...

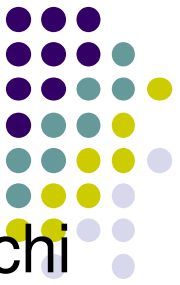
BUILD SUCCESSFUL (total time: 6 seconds)

try



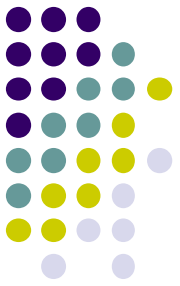
- Si noti come il codice del metodo main nell'esempio sia stato leggermente modificato per sfruttare il meccanismo secondo il quale, a seguito del verificarsi di una eccezione, le istruzioni successive alla chiamata del metodo che ha sollevato l'eccezione non vengono eseguite

catch()



- Uno stesso blocco try può essere seguito da più blocchi catch
 - Il parametro tra parentesi è chiamato parametro eccezione e specifica il TIPO di eccezione che quel blocco catch può gestire
 - Quando si verifica una eccezione in un blocco try viene eseguito il primo (lessicograficamente) blocco catch associato al try il cui tipo **corrisponde** al tipo dell'eccezione
- L'eccezione specificata può essere utilizzata all'interno dell'handler, ad esempio chiamando implicitamente il metodo toString per l'eccezione come nell'esempio DivisionePerZero

Multi-catch

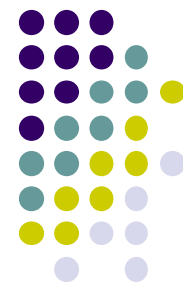


- Se i corpi dei diversi blocchi catch relativi ad uno stesso costrutto try sono identici, si possono «fattorizzare» in un unico catch:

`catch(Tipo1 | Tipo2 | Tipo3 e)`

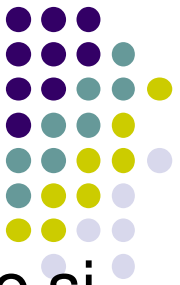
Si può specificare un qualunque numero di tipi

Eccezioni non rilevate



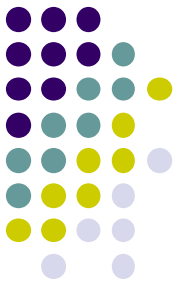
- Una eccezione non rilevata è una eccezione che si verifica e per la quale non sia presente un blocco catch in grado di gestirla
 - In questo caso l'eccezione si propaga lungo la catena delle chiamate fino ad arrivare la metodo main provocando la terminazione del programma (come abbiamo visto nel primo caso della DivisionePerZero)
 - Un caso più complesso si presenta se il programma è ***multithread***, in questo corso trattiamo programmi che hanno un unico thread di esecuzione

Stack Unwinding



- Nel caso in cui la eccezione sollevata da un metodo si propaga lungo la catena delle chiamate perché non incontra mai un blocco catch in grado di «bloccarla» e di gestirla, tutti i metodi della catena terminano
 - Questo provoca la de-allocazione dei loro record di attivazione e il conseguente **scaricamento dello stack** (stack unwinding)
- Quando l'eccezione arriva al metodo main, se il main non è in grado di gestire l'eccezione anche esso viene terminato

Esempio



```
public class stackunwinding {  
  
    public static void main(String[] args) {  
        funzione1();  
    }  
  
    public static void funzione1() {  
        funzione2();  
    }  
  
    public static void funzione2() {  
        funzione3();  
    }  
  
    public static void funzione3() {  
        throw new IllegalArgumentException("Errore!");  
    }  
  
}
```

Esecuzione



run:

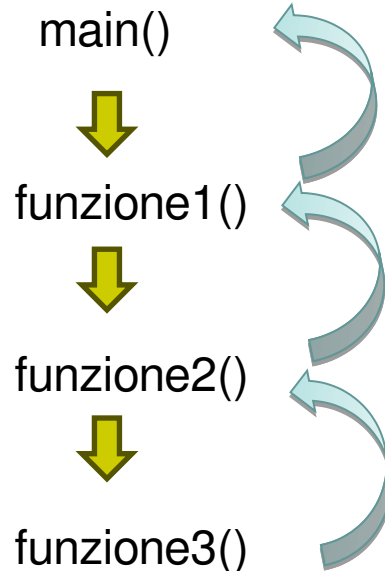
```
Exception in thread "main" java.lang.IllegalArgumentException: Errore!  
    at eccezioni.stackunwinding.funzione3(stackunwinding.java:26)  
    at eccezioni.stackunwinding.funzione2(stackunwinding.java:22)  
    at eccezioni.stackunwinding.funzione1(stackunwinding.java:18)  
    at eccezioni.stackunwinding.main(stackunwinding.java:14)
```

[/Users/valeria/Library/Caches/NetBeans/14/executor-snippets/run.xml:111](#): The following error

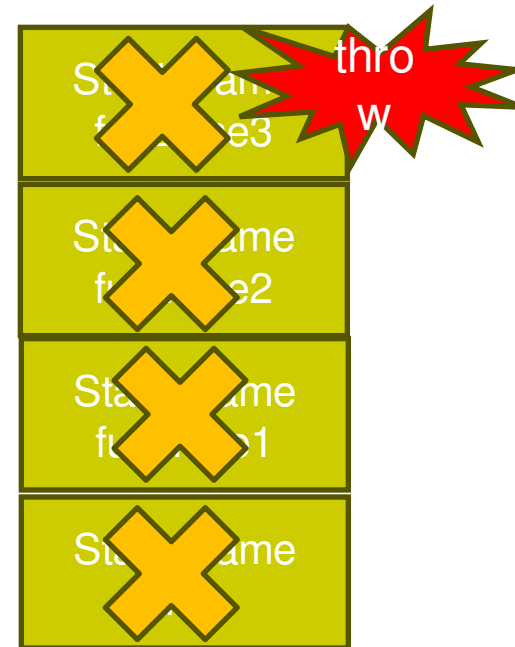
[/Users/valeria/Library/Caches/NetBeans/14/executor-snippets/run.xml:68](#): Java returned: 1

BUILD FAILED (total time: 0 seconds)

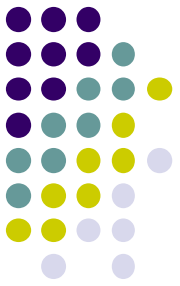
Catena delle chiamate:



Propagazione della eccezione



Rilancio di un'eccezione

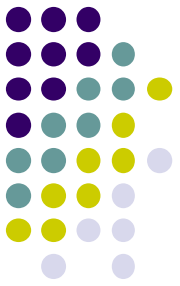


- Se un handler non è in grado di gestire completamente un'eccezione, può *rilanciarla*, assumendo che un altro handler possa completarne la gestione

```
try {  
    // codice che può lanciare un'eccezione di tipo E  
}  
catch (E e) {  
    // fa qualcosa  
    // rilancia l'eccezione originaria  
    throw e;  
}
```

```
public static void funzione2() {  
    try {  
        funzione3();  
    }  
    catch(IllegalArgumentException e){  
        //codice per una parziale gestione  
        throw e;  
    }  
}
```

Specificazione delle eccezioni



- L'interfaccia di un metodo può specificare la lista di eccezioni che possono essere lanciate da tale metodo

```
public static int DivisionePerZero(int a, int b) throws  
    ArithmeticException { // codice }
```

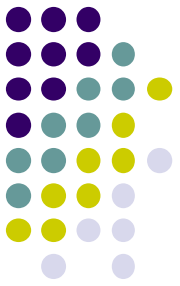
- La clausola **throws** può contenere una lista di eccezioni separate da virgola
- Il vantaggio è che la dichiarazione appartiene ad un'interfaccia che è visibile al chiamante

Quando NON usare la gestione delle eccezioni

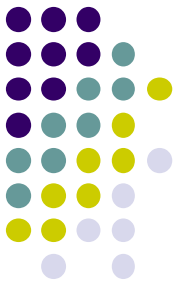


- La gestione delle eccezioni fornisce un meccanismo potente per il controllo degli errori che si possono verificare a tempo di esecuzione, tuttavia non dovrebbe essere usata:
 - In tutti quei casi in cui è possibile utilizzare il normale controllo di flusso (quindi ad esempio un costrutto condizionale e variabili booleane)
 - Per gestire problemi causati da eventi asincroni al programma (ad esempio l'arrivo di un messaggio dalla rete, il click del mouse, etc.)
- E' bene tenere sempre presente inoltre che una eccezione non gestita può provocare la terminazione del programma (e che la legge di Murphy è sempre in agguato!)

Vantaggi della gestione delle eccezioni



- Il programmatore ha il controllo sulla gestione delle situazioni eccezionali, decidendo in modo flessibile come trattarle.
- Ogni condizione anomala deve essere affrontata: situazioni non gestite causano la terminazione del programma.
- Le istruzioni per la gestione delle eccezioni sono distinte da quelle del flusso normale del programma, mantenendo la struttura complessiva del codice più chiara.



Riferimenti

- Programmare in Java
 - Cap. 11 fino al §11.4, stack unwinding §11.17