

Interfacce

Interfacce

- Un'interfaccia è una collezione di *metodi* - privi di implementazione - e di *costanti*
 - E' come una classe in cui tutti i metodi sono astratti
- Se una classe *implementa* una data interfaccia, il compilatore verifica che nella classe esistano le definizioni per tutti metodi definiti nell'interfaccia



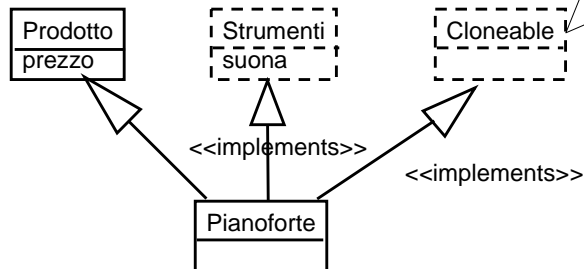
Interfacce: un esempio

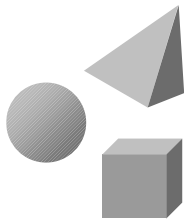
```
interface strumenti {  
    String descrizione = "strumento musicale";  
    // valore implicitamente static e final  
  
    void suona() ;  
    // notare l'assenza della implementazione  
}  
  
class flauto implements strumenti {  
    public void suona() {  
        System.out.println(" fiuuuuuuu ");  
    }  
  
    public String descrizione() {  
        return descrizione + " flauto";  
    }  
}
```



Ereditarietà multipla tramite le interfacce

Interfaccia definita nella libreria standard di Java che deve essere implementata dalle classi che consentono l'uso della clone





Le eccezioni

Le eccezioni

- È un meccanismo per la gestione degli errori in un programma
 - in C solitamente è il codice ritornato da una funzione
- Consente di separare il flusso normale di esecuzione da situazioni inaspettate
- Una condizione eccezionale è un problema che non consente la continuazione dell'esecuzione del metodo corrente
 - Nel contesto corrente non ci sono informazioni sufficienti per correggere l'errore e continuare l'esecuzione
 - Esempio: si verifica una divisione per zero



La gestione delle eccezioni

- Una eccezione viene “sollevata” in un certo contesto (**throw**)
- Il contesto corrente di esecuzione viene interrotto
- L'*exception handler* si occupa di cercare un nuovo contesto in grado di gestire l'eccezione, altrimenti l'errore viene passato al supporto run time e il programma termina



Un esempio

- In un sottoprogramma...

```
if (t == null)
    throw new NullPointerException();
.....
```
- Nel chiamante...

```
try {
    p(ref);
} catch (NullPointerException e) {
    //operazioni di gestione dell'eccezione
    System.out.println("Errore e");
}
```



Dichiarazione delle eccezioni

- Un sottoprogramma deve dichiarare le eccezioni che solleverà
- Per esempio:

```
void p() throws MyException {  
    .....  
    throw new MyException();  
    .....  
}
```
- la correttezza della gestione delle eccezioni è verificata a *compile time*



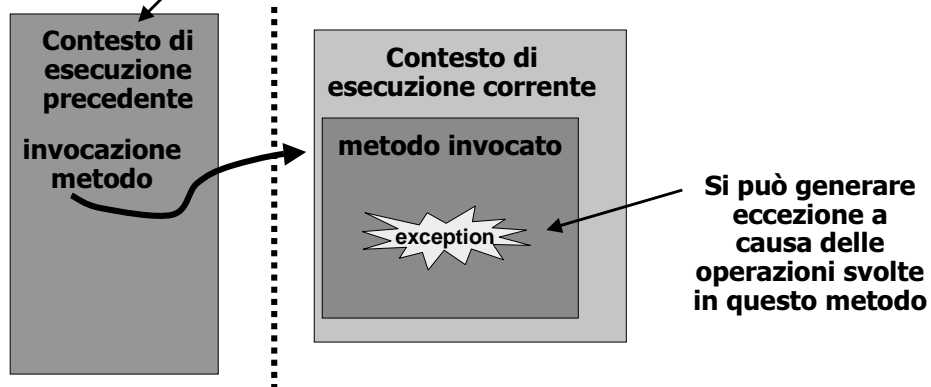
Comportamento in seguito al verificarsi di eccezioni

- Il chiamante di un metodo che genera eccezioni può avere due comportamenti:
 - ignorare le eccezioni, lasciandone la gestione al proprio chiamante
 - cercare di gestire l'eccezione
- Nel primo caso il chiamante deve segnalare con la clausola **throws** nella propria intestazione che può generare (trasmettere) eccezioni.
- Nel secondo caso utilizza il costrutto **try-catch** per eseguire porzioni di codice in modo controllato

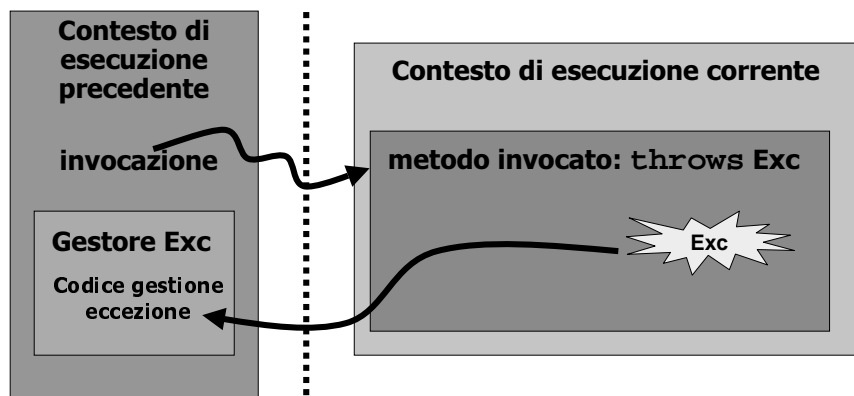


Eccezioni: comportamenti possibili

L'eccezione è associata al concetto di "programmazione per contratto": un contesto invoca un metodo che viene eseguito in un altro contesto

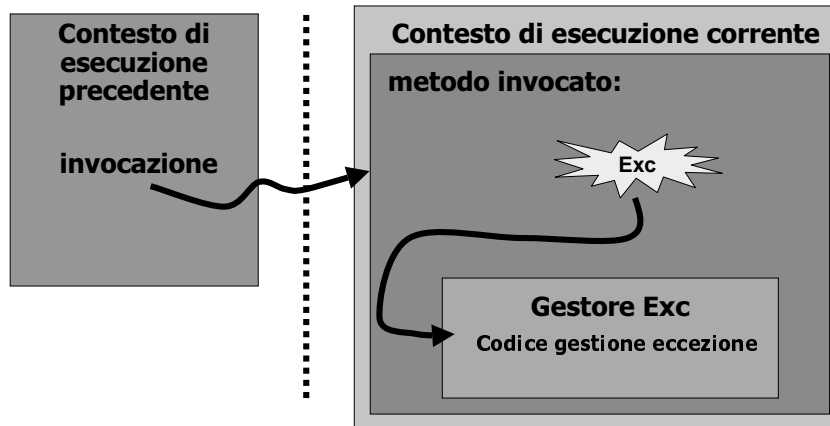


Restituire le eccezioni (dichiararle)



- È la clausola **throws** che consente al compilatore di "lasciare il compito" di gestione dell'eccezione a qualcun altro

Gestire le eccezioni (catturarle)



- Sono le clausole **try-catch** che consentono al compilatore di considerare l'eccezione gestita nello stesso contesto



Eccezioni ignorate

```
class EsempioB {  
    static void metodoInutile() throws IllegalAccessException {  
        System.out.println("metodoInutile");  
        throw new IllegalAccessException("prova");  
    }  
    public static void main (String[] arg) {  
        EsempioB.metodoInutile();  
    }  
}
```

The `main` method call `EsempioB.metodoInutile();` is circled in red, and an arrow points to it from the text below.

- È corretta la dichiarazione del metodo ma non il main: deve esser gestita nel chiamante l'eccezione sollevabile dal metodo chiamato

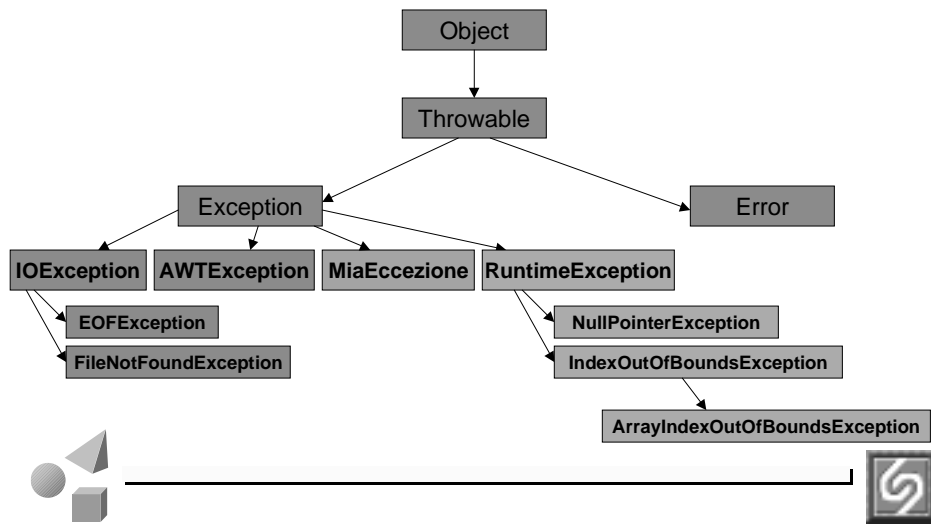
EsempioB.java:7: Exception java.lang.IllegalAccessException must be caught, or it must be declared in the throws clause of this method.

```
EsempioB.metodoInutile();  
                        ^
```



Gerarchia di Eccezioni

- Gerarchia predefinita ma estendibile



Gerarchia di eccezioni

- Nella superclasse **Throwable** sono definiti i metodi utilizzati dalla classe **Exception**
 - **getMessage()** e **toString()**
 - **printStackTrace()**
- Catturare selettivamente le eccezioni:
 - le classi più specifiche contengono maggiore informazione sul tipo di eccezione che si è verificata
 - ▲ `catch (FileNotFoundException ecc_particolare) { ... }`
 - ▲ `catch (IOException ecc_generale) { ... }`
 - polimorfismo: `catch (Exception e)` dovrebbe essere l'ultima `catch` di un'eventuale sequenza di `catch` più specifiche
 - ▲ altrimenti il compilatore segnala che altre clausole non possono essere raggiunte (*catch not reached*)

Gerarchia di eccezioni (2)

- **Exception** appartiene al package **java.lang**, ma le eccezioni possono appartenere a diversi package (**java.io.IOException** per tutte quelle di accesso agli stream)
- È possibile (utile e necessario) definire nuove eccezioni per gestire particolari situazioni
 - class **MiaEccezione** extends **Exception** { ... }
- È possibile realizzare **try - catch** annidate
 - ripetere le operazioni in diverse situazioni, dopo aver ripetuto certe operazioni



RuntimeException

- Particolare insieme di eccezioni che possono essere sempre sollevate da qualsiasi porzione di codice
- Non devono essere gestite dall'utente: è compito del supporto *run time*
 - Comportamento di default: se una **RuntimeException** esce da un **main** senza essere catturata, allora viene invocato **PrintStackTrace()** e il programma termina
 - Sollevare la seguente eccezione è superfluo

```
if (t == null)
    throw new NullPointerException();
```



Clausola Finally

- codice che deve essere eseguito comunque, indipendentemente dal fatto che si sia verificata un'eccezione (ad esempio per chiudere un file)

```
try {  
    codice che può sollevare eccezioni  
}  
catch (TipoEccezione1 VariabileEccezione1) {  
    gestione dell'eccezione1  
}  
...  
catch (TipoEccezioneN VariabileEccezioneN) {  
    gestione dell'eccezioneN  
}  
finally {  
    codice da eseguire comunque  
}
```

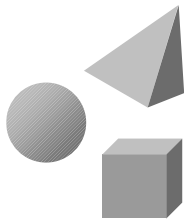


Vantaggi della gestione delle eccezioni

- Viene separata la gestione degli errori dal codice normale
- Si propagano gli errori sullo stack, gestendo livelli differenti di astrazione diversi
- Gli errori vengono raggruppati e differenziati per tipologie
- Si garantisce che eventuali malfunzionamenti non passino inosservati per negligenza del programmatore.

➔ **Risultato: Codice molto più robusto**





I packages

Riutilizzo del codice

- Librerie: “Separare le cose che cambiano da quelle che rimangono uguali”
- Organizzazione del codice esistente
- Specificare cosa è accessibile al cliente della libreria (*access specifier*):
 - **public**
 - **friendly**
 - **protected**
 - **private**



Package

- Clausola **import** per accedere alle classi di in un package:
 - Utilizzo della classe **Vector** che si trova nel package java.util
 - ▲ `java.util.Vector mioVettore = new java.util.Vector();`
 - oppure:
 - ▲ `import java.util.Vector;`
 - ▲ `Vector mioVettore = new Vector();`
 - Accedere a tutte le classi del package (import multiplo):
 - ▲ `import java.util.*;`
 - Convenzione: nomi package sempre minuscoli
 - Classi in package diversi possono avere lo stesso nome



Definizione di un package

- Definire come prima istruzione di una compilation unit l'appartenenza al package:

```
package mialibreria;  
public class MiaClasse {  
    ...  
}
```

- Per utilizzare la classe **MiaClasse**:

- 1) `miaLibreria.MiaClasse m = new miaLibreria.MiaClasse();`
- 2)

```
import mialibreria.*;  
MiaClasse m = new MiaClasse();
```



Memorizzazione dei package su file system ed esecuzione

- In genere le classi appartenenti ad un package vengono memorizzate in una directory che ha il nome del package
 - C:\users\Elisabetta\didattica\SE-Como\esempi\miaLibreria
- In fase di esecuzione è necessario specificare il nome del package

```
java miaLibreria.MiaClasse
```



Java API Packages

- package java.applet
- package java.awt - Abstract Window Toolkit
- package java.beans
- package **java.io** - gestione stream input output
- package **java.lang** - tipi di dati, thread, exception
- package java.math
- package java.net - socket e URL
- package java.rmi - Remote Method Invocation
- package java.security
- package java.sql
- package **java.util** - hashtable, vector, sting tokenizer



Specificatori di accesso

