UNIVERSITÀ DEGLI STUDI "ROMA TRE"
CORSO DI STUDI IN SCIENZE COMPUTAZIONALI
IN490 - LINGUAGGI DI PROGRAMMAZIONE – A.A. 2017-2018

M. PEDICINI

ESONERO DEL 13/04/2018 – TEMPO 2H00

Esercizio 1. *(1) Si descriva il costrutto delle* eccezioni *in un linguaggio di programmazione.*

Soluzione. L'eccezione è un costrutto che permette di modificare il normale flusso di esecuzione di un programma. Normalmente un programma esegue le istruzioni in sequenza e tramite alcuni costrutti per il controllo del flusso di esecuzione si può gestire il flusso ed eseguire parti diverse del programma: l'esempio base è l'istruzione `if c then A else B` che permette in base al valore della condizione di eseguire alternativamente la parte `A` o quella `B`.

Nei linguaggi più moderni i costrutti per il controllo del flusso sono sempre strutturati (salti condizionati o cicli, soggetti ad un test che ne condiziona il comportamento), talvolta è necessario gestire degli eventi che richiedono la modifica del normale controllo del flusso di esecuzione; il metodo per gestire queste situazioni è fornito dal costrutto delle eccezioni che introduce la possibilità di forzare il controllo del flusso a "saltare" verso una procedura in grado di gestire l'evento speciale.

L'esempio tipico di eccezione si verifica quando c'è un problema con la gestione del flusso a run-time e l'esecuzione di un programma viene interrotta (si pensi al caso di una violazione del segmento della memoria assegnato al processo). In questo caso l'eccezione modifica il flusso dell'esecuzione passando direttamente all'istruzione che termina l'esecuzione. In generale, non è l'effetto voluto, perchè sarebbe preferibile che al verificarsi dell'evento che deve modificare il controllo del flusso si passi ad una parte del codice specificatamente dedicata alla gestione della situazione.

Questo effetto si ottiene specificando che il controllo deve spostarsi in una diversa procedura (istruzione `throw <nome eccezione>`) e la procedura in cui proseguire con l'esecuzione viene individuata da un'opportuna istruzione seguita dal nome dell'eccezione (istruzione `try <blocco che puo' generare eccezioni> catch <nome eccezione> <blocco da eseguire>`).

L'interruzione del normale flusso di esecuzione comporta l'uscita da ogni procedura/funzione fino a quando l'istruzione `try` non viene incontrata, a questo punto si può verificare che nella corrispondente `catch` abbiamo il nome dell'eccezione ed allora il controllo passa il blocco della `catch` oppure la `catch` non tratta l'eccezione in oggetto ed allora l'eccezione continua a risalire tutti i blocchi fino eventualmente a quello principale, che provoca l'arresto dell'esecuzione (*unchecked exception*).

□

(2) *Si descriva l'output stampato dal seguente frammento in cui il passaggio dei parametri avviene per* valore - risultato*:*

```
 1 int x = 2;
 2 int w = 6;
 3 int z = 11;
 4
 5 int f (int y){
 6 y = 21;
 7 throw E;
 8 return (x++)+y;
 9 }
10
11 int g (int y){
12  y = 100;
13  try { throw E; } catch E {}
14  return (x++) + y;
15 }
16
17 try { f(w); } catch E {} print(x,w,z);
18 z = g(w);
19 print(x,w,z);
```

Soluzione. Per prima cosa osserviamo che l'eccezione `E` non è dichiarata nel sorgente fornito (quindi supponiamo che sia dichiarata altrove) e che tutte le occorrenze di `E` ricadano nello scope della stessa dichiarazione. Dopo aver eseguito le prime righe avremo un ambiente $\sigma_3 = \{x \leftarrow 2, w \leftarrow 6, z \leftarrow 11\}$ a questo punto si esegue alla linea 17 il `try` e in particolare si passa alla procedure `f`.

Poiche lo stile di questo linguaggio per il passaggio dei parametri è quello *valore-risultato* si esegue il blocco alla linea 5 con l'ambiente $\sigma_4 = \{x \leftarrow 2, w \leftarrow 6, z \leftarrow 11, y \leftarrow 6\}$ dove y è una variabile locale che è stata valorizzata con il valore del parametro w. $\sigma_5 = \{x \leftarrow 2, w \leftarrow 6, z \leftarrow 11, y \leftarrow 21\}$ a questo punto viene eseguita l'istruzione `throw E`: il controllo esegue un'uscita dal blocco corrente (senza eseguire la return) e torna alla linea 17, con l'ambiente corrispondente $\sigma_6 = \{x \leftarrow 2, w \leftarrow 6, z \leftarrow 11\}$, l'eccezione è intercettata dalla `catch` con blocco vuoto e dunque poi l'esecuzione riprende con la `print` a linea 17.

Il programma stampa i valori delle tre variabili che sono `2 6 11`.

Si passa ad eseguire la linea 18, invocando l'esecuzione della funzione `g`. $\sigma_7 = \{x \leftarrow 2, w \leftarrow 6, z \leftarrow 11, y \leftarrow 6\}$ al passo successivo eseguendo la linea 12 l'ambiente viene modificato nella componente `y`: $\sigma_8 = \{x \leftarrow 2, w \leftarrow 6, z \leftarrow 11, y \leftarrow 100\}$ la linea 12 solleva l'eccezione `E` che però viene subito intercettata dalla `catch` corrispondente (sempre a linea 13), che passa il controllo al blocco vuoto a linea 13, si continua l'esecuzione del blocco 14 e viene restituito il valore 103 risultato della valutazione di `(x++) + y` la quale fa ritornare il controllo alla linea 18 con l'ambiente $\sigma_9 = \{x \leftarrow 3, w \leftarrow 100, z \leftarrow 103\}$ in cui il valore delle tre variabili viene aggiornato: quello della `x` per effetto dell'istruzione `x++` eseguita alla linea 14, quello della w per effetto del passaggio dei parametri *valore-risultato* per cui la variabile locale `y` nel blocco della funzione `g` assume valore 100, che viene copiato come risultato nel parametro della chiamata `w` e l'assegnazione a linea 18 modifica il valore di `z`.

Infine, alla linea 19, il programma stampa i valori delle tre variabili che sono `3 100 103`.

□

Esercizio 2. *(1) Spiegare in cosa consiste in un linguaggio ad oggetti la* selezione dinamica *dei metodi (cioè nel quale tutti i metodi sono "virtual member functions", nella terminologia C++).*

Soluzione. Nella programmazione object oriented uno dei meccanismi base è quello dell'ereditarietà, che permette di definire una nuova classe come estensione della definizione di un'altra classe. Nel meccanismo dell'ereditarietà gli attributi e i metodi della sovraclasse sono ereditati dalla sottoclasse. Il programmatore ha la facoltà di definire attributi e metodi della sottoclasse con lo stesso nome e tipo diverso (*overloading*) oppure stesso nome e stesso tipo. Quando un attributo con stesso nome e tipo viene ridefinito nella sottoclasse l'attributo originario viene mascherato (*shadowing*) e, benché allocato, non è più accessibile come elemento dell'oggetto corrente, quando è un metodo ad essere ridefinito allora si parla di *overriding*. Nel linguaggio C++ un metodo della sovraclasse può essere ridefinito nella sottoclasse se è specificato come "virtual". Nei linguaggi con *selezione dinamica* dei metodi (come Java), l'overriding è abilitato per default per tutti i metodi ed è disabilitato quando il metodo viene dichiarato `private`. L'effetto della selezione dinamica dei metodi è che al momento di eseguire una chiamata a funzione per un oggetto di una classe viene invocato il primo metodo accessibile che corrisponde al tipo di oggetto, quindi in generale il metodo ridefinito nella sottoclasse.

Dunque dato un certo oggetto o, quando viene invocato un metodo m non è dato sapere staticamente quale metodo debba essere invocato poichè questo dipende dalla definizione dell'oggetto concreto o, mentre il compilatore accetta l'invocazione se il tipo è rispettato.

Se il metodo m è stato ridefinito allora vi possono essere più versioni di m tutte applicabili per o. In fase di esecuzione, viene selezionata l'implementazione di m utilizzata in un'invocazione

```
o.m(parametri)
```

ed è determinata dal tipo dell'oggetto o. La soluzione dell'ambiguità di quale metodo invocare viene risolta a run-time sfruttando il tipo "dinamico" dell'oggetto ossia il tipo del riferimento all'oggetto (indipendentemente dalla classe di cui è istanza).

□

(2) Si considerino le seguenti classi

```
class A {
int x = 7;

int foo () {return g();}
int g() {return x;}
}

class B extending A{
int g() {return 2*x;}
}
```

ed il seguente frammento di codice:

```
B b = new B();
A a = b;
int zz = a.foo()+ b.x ;
```

Si dica qual è il valore di `zz` *al termine dell'esecuzione del frammento.*

Soluzione. In questo esempio viene creato un solo oggetto come istanza della sottoclasse `B`. Abbiamo poi due variabili `b` di tipo `B` ed `a` di tipo `A` che fanno riferimento a questo oggetto.

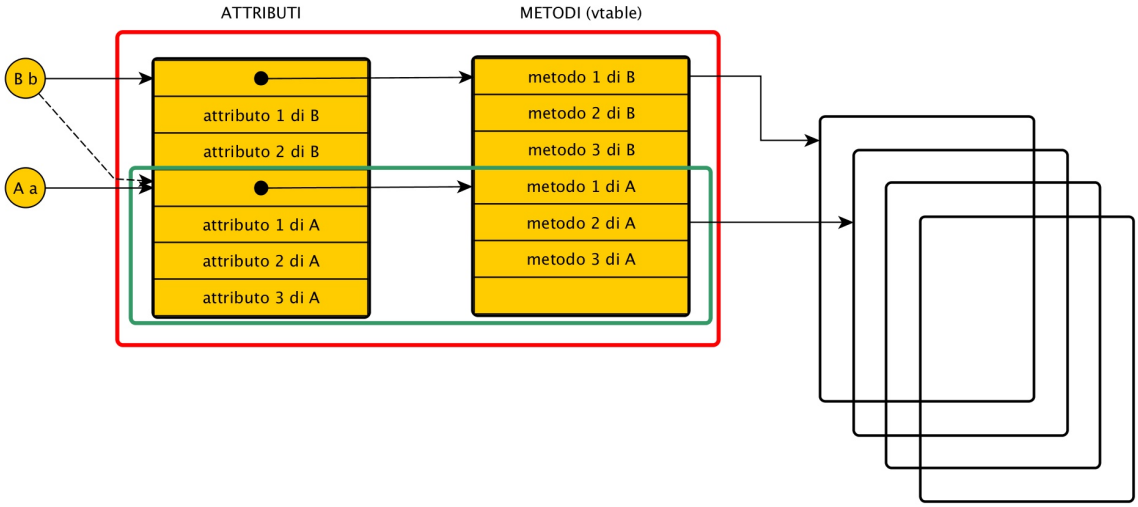
Quando il viene assegnata la variabile `zz` viene invocato il metodo `a.foo()` l'oggetto è istanza della classe `B` per cui nella vista dei metodi troviamo il metodo `foo` dichiarato nella classe `A` questo a sua volta invoca il metodo `g()` e poichè l'oggetto è della classe `B` questa volta è il metodo ridefinito in `B` ad essere selezionato dinamicamente.

Poichè `g()` restituisce il doppio del valore dell'attributo `x` abbiamo che il risultato è che `zz` vale `2*7+7 = 21`.

□

Esercizio 3. *(1) Spiegare con un diagramma la struttura dati che viene utilizzata per rappresentare in memoria la struttura di una classe (*virtual tables*);*

Soluzione. Al momento di creare l'istanza di una classe `B` che estende un'altra classe `A` bisogna che venga allocata memoria per l'oggetto di tipo `A` e per quello di tipo `B`, bisogna inoltre allocare una tabella in cui appaiano i metodi specificati nelle due classi e nel caso di overriding bisogna che siano tenuti in considerazione solo i metodi ridefiniti.



□

(2) Si dia la struttura delle `vtable` *relative alle seguenti quattro classi:*

```
class A{
int x = 0;
void f() {x=12;}
}

class B{
int x = 0;
void f() {x=12;}
}

class C extending A {
int x = 13;
void g(){x=100;}
}

class D extending C {
void g(){x=200;}
}
```