Corso di laurea in Comunicazione digitale — Programmazione I
AA 2009/10 — Prof. M. A. Alberti

I Appello - 26 gennaio 2010

Cognome **Nome**

Matricola

1.

Date le classi `BevandaAlcolica` e la sottoclasse `Vino` :

```
class BevandaAlcolica {
      private int gradazione;
      private String colore;
      private String nome;
      BevandaAlcolica (int g, String c, String n){
            gradazione = g;
            colore = c;
            nome = n;
      }
      public int incr_gr(){
            return ++gradazione;
      }
      public int get_gr(){
            return gradazione;
      }
      public int set_gr(int g){
            return gradazione = g;
      }
      public String toString() {
            return "bevanda alcolica " + nome + " " + colore +
                   ", gradi " + gradazione + ". ";
      }
}
class Vino extends BevandaAlcolica{
      private int gradazione;
      private String nome;
      Vino(int g, String n, String c){
            super(g/2, c, "");
            nome = n;
            gradazione = g;
      }
      public int get_gr(){
            return gradazione;
      }
      public int metodo_speciale(){
            return super.incr_gr();
      }
      public String toString(){
            return super.toString()+" " + nome +
                              ", gradi " + gradazione;
      }
}
```

Dire se la compilazione ha successo. **SI** NO
Dire se è possibile definire campi con lo stesso nome nelle due classi (in particolare in queste si usa l'identificatore `nome` e `gradazione` in entrambe). **SI** NO
Dire come si chiama questo meccanismo: **oscuramento o shadowing**

Dire quali membri eredita la classe `Vino` dalla superclasse: **nessun campo ma tutti i metodi `(incr_gr(), set_gr(), get_gr(), toString())` taluni dei quali saranno oscurati**

I metodi `get_gr()` e `toString()` sono definiti per entrambe le classi. Dire come si chiama questo meccanismo: **sovrascrittura**

Supponete di avere dichiarato:

```
BevandaAlcolica aperitivo =
                new BevandaAlcolica(5, "bianca", "sangria");
```

Valutate le seguenti espressioni in sequenza:

`out.println(aperitivo)` **bevanda alcolica sangria bianca, gradi 5.**	
`aperitivo.incr_gr()`	**6**
`aperitivo.set_gr(aperitivo.incr_gr())`	**7**
`aperitivo.incr_gr()`	**8**
`aperitivo.get_gr()`	**8**
`out.println(aperitivo)` **bevanda alcolica sangria bianca, gradi 8.**	

Ora supponete di aver dichiarato:

```
BevandaAlcolica bicchiere = new Vino(12, "cabernet", "rosssa")
```

E' lecito o errore? **Lecito** Errore
Perché? **Ogni oggetto della classe `Vino`, estensione della classe `BevandaAlcolica` appartiene anche alla suddetta classe.**

Valutate le seguenti espressioni in sequenza:

`out.println(bicchiere)` **bevanda alcolica rossa, gradi 6. cabernet, gradi 12**	
`bicchiere.incr_gr()`	**7**
`bicchiere.get_gr()`	**12**

Scrivete l'espressione per richiamare il metodo `metodo_speciale()`
`((Vino)bicchiere).metodo_speciale()`, quindi valutatela: **8**

`bicchiere.get_gr()`	**12**
`bicchiere.set_gr(bicchiere.get_gr()+1)`	**13**
`bicchiere.get_gr()`	**12**
`out.println(bicchiere)` **bevanda alcolica rossa, gradi 13. cabernet, gradi 12**	

2.

Dato il metodo `int f(int, int, int)` seguente:

```
int f(int n, int f0, int f1)  {
  if (n==0)
    return f0;
  else if (n==1)
    return f1;
  else
    return (f(n-1, f1, f0+f1)); // indirizzo di rientro A
}
```

tracciare lo stato dello stack delle chiamate dei metodi durante l'esecuzione fi f(6, 0, 1). Si ricordi che rit indica l'indirizzo di rientro e val il valore di rientro di un metodo chiamato.

n	6	5	4	3	2	1			
f0	0	1	1	2	3	5			
f1	1	1	2	3	5	8			
val	~~?~~ 8	~~?~~ 8	~~?~~ 8	~~?~~ 8	~~?~~ 8	?			
rit	~~?~~ A	~~?~~ A	~~?~~ A	~~?~~ A	~~?~~ A	?			
	record 1	record 2	…						

Quale valore riporta la chiamata di `f(6, 0, 1)`? **8**

Quante chiamate ricorsive vengono effettuate? **6, se includiamo nel conto anche la prima**

3.

Scrivere un metodo (prototipo e corpo) per computare **iterativamente** la funzione di Fibonacci. Allo scopo sarà necessario utilizzare un ciclo che coinvolge tre variabili, che possiamo definire e inizializzare nel modo seguente:

```
int precprec=0, prec=1, temp=0;
```

Ricordiamo che la funzione di Fibonacci è definita per numeri interi naturali nel seguente modo:

```
        |  0 se n = 0
f(n) =  |  1 se n = 1
        |  f(n-1) + f(n-2) se n > 1
```

```
public int f(int n){
            int prec=1, precprec=0, tmp=0;
            for (int i=0; i<=n; i++){
                        tmp += precprec; // 0, 1, 1, 2, 3, 5, 8
                        precprec = prec; // 1, 0, 1, 1, 2, 3, 5
                        prec = tmp;      // 0, 1, 1, 2, 3, 5, 8
            }
            return tmp;
      }
}
```

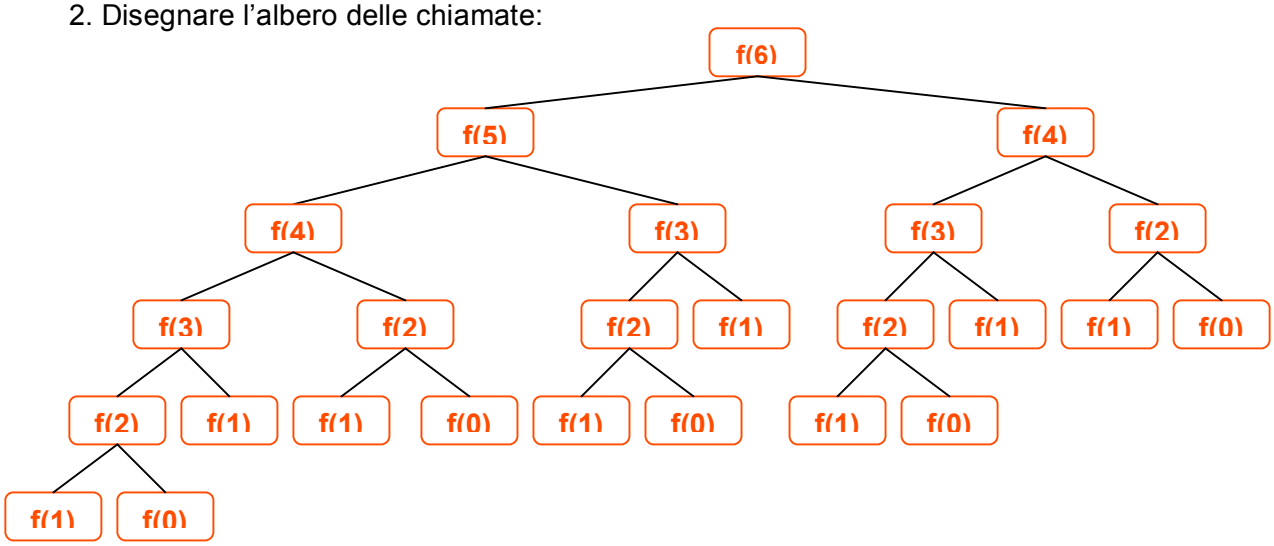
4.

Dato il metodo `int f(int)` che computa la funzione di Fibonacci in modo ricorsivo:

```
int f(n) {
if (n==0 || n== 1)
      return n;
else
```

```
        return (f(n-1) + f(n-2));
    }
```

1. Calcolare il numero delle chiamate ricorsive necessarie per calcolare `f(6)`? **25**
2. Disegnare l'albero delle chiamate:



5.

Stabilire la relazione tra quest'ultimo metodo e il metodo dell'esercizio 2.
I due metodi calcolano la stessa funzione.

Commentare. ? **Il metodo dell'esercizio 2 è più efficiente. Infatti per calcolare il numero di Fibonacci `f(6)` il metodo dell'esercizio 2 effettua 6 chiamate ricorsive, mentre il metodo dell'esercizio 4 ne effettua 25.**

6.

Considerate le classi `A` e `B`, estensione della classe `A`:

```
class A {
   protected codice;
   protected String tipo;
   A(int c, String t) {
      codice = c;
      tipo = t;
   }
}
class B extends A {
   private int codice;
   B (int c) {
      super(c-1, "esteso");
      codice = c;
   }
}
```

1. Definite il costruttore della classe `B`, estensione della classe `A`, rispettando i seguenti requisiti:
   a. ogni oggetto di classe `B` è caratterizzato da un numero di codice
   b. il suo antenato ha per convenzione come campo `codice` lo stesso codice –1 e come campo `tipo` il valore "`esteso`".

2. Osservate i due membri `codice` definiti nelle due classi. Il membro `codice` di classe `A` dichiarato `protected` viene ereditato dalla classe `B`? **SI** NO

3. Si dice che il campo `codice` di classe `B` **oscura** quello di classe `A`

4. Dire se è corretta la definizione seguente di un metodo di classe `B`: **SI** NO
```
int get_codice() { return super.codice + codice; }
```

5. Definita nella classe `A` un metodo `main` con le istruzioni seguenti, valutate le seguenti espressioni:
```
A a = new A(21, "non esteso");
B b = new B(222);
out.println(a.codice);          21
out.println(a.tipo);        non esteso
out.println(b.get_codice());   443
out.println(b.tipo);          esteso
```

6. Tra le istruzioni precedenti, potremmo sostituire l'espressione `b.get_codice()` con l'espressione `b.codice`? SI **NO**

7. **senza considerare che il metodo `get_codice()` riporta la somma dei due campi `codice`, il metodo non potrebbe essere sostituito dall'espressione `b.codice` perchè il campo `codice` della classe `B` è dichiarato private e quindi non è accessibile alla classe `A`**