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 in entrambe). SI NO
Dire come si chiama questo meccanismo:

Dire quali membri eredita la classe `Vino` dalla superclasse:

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

Supponete di avere dichiarato:

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

Valutate le seguenti espressioni in sequenza:

`out.println(aperitivo)`	
`aperitivo.incr_gr()`	
`aperitivo.set_gr(aperitivo.incr_gr())`	
`aperitivo.incr_gr()`	
`aperitivo.get_gr()`	
`out.println(aperitivo)`	

Ora supponete di aver dichiarato:

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

E' lecito o errore? Lecito Errore
Perché?

Valutate le seguenti espressioni in sequenza:

`out.println(bicchiere)`	
`bicchiere.incr_gr()`	
`bicchiere.get_gr()`	

Scrivete l'espressione per richiamare il metodo `metodo_speciale()`, quindi valutatela:

`bicchiere.get_gr()`	
`bicchiere.set_gr(bicchiere.get_gr()+1)`	
`bicchiere.get_gr()`	
`out.println(bicchiere)`	

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));
}
```

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									
f0									
f1									
val									
rit									

record 1 record 2 …

Quale valore riporta la chiamata di `f(6, 0, 1)`?
Quante chiamate ricorsive vengono effettuate?

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

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
```

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)`?
2. Disegnare l'albero delle chiamate

5.
Stabilire la relazione tra quest'ultimo metodo e il metodo dell'esercizio 2.

Commentare.

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) {




   }
}
```

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 membro `codice` di classe `B`
   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);
   out.println(a.tipo);
   out.println(b.get_codice());
   out.println(b.tipo);
   ```
6. Tra le istruzioni precedenti, potremmo sostituire l'espressione `b.get_codice()` con l'espressione `b.codice`? SI NO