Correzione Esonero del corso di Programmazione a Oggetti

Roma, 26 novembre 2008

Domanda 1 Quale delle seguenti affermazioni è vera:
☐ Il tipo `OnorevoleMaggioranza` è sottotipo di `OnorevoleOpposizione`;
☑ Il tipo `OnorevoleMaggioranza` è sottotipo di `Politico`;[1]
☐ Il tipo `Politico` è sottotipo di `Onorevole`.

Domanda 2 Quale delle seguenti affermazioni è falsa:
☐ Il tipo `OnorevoleMaggioranza` è sottotipo di `Onorevole`;
☐ Il tipo `OnorevoleOpposizione` è sottotipo di `Politico`;
☑ Il tipo `FrancoTiratore` è sottotipo di `Ministro`.[2]

Domanda 3 L'istruzione `Politico xy = new FrancoTiratore();`
☑ compila ed esegue correttamente;
☐ compila, ma dà un errore in esecuzione perchè la classe `FrancoTiratore` non ha il costruttore;
☐ dà errore in compilazione.

Domanda 4 L'istruzione `Politico xy = new FrancoTiratore();`
☐ dà errore in compilazione;
☐ non produce nessun output;
☑ stamperà `Onorevole silenzioso creato.`[3].

Domanda 5 L'istruzione `FrancoTiratore xy = new OnorevoleMaggioranza();`
☑ dà errore in compilazione;[4]
☐ dà errore in esecuzione;
☐ stamperà `Onorevole silenzioso creato`.

[1] Infatti la relazione di sottotipaggio è transitiva.
[2] Si tratta di classi "sorelle".
[3] osservate che nella classe `FrancoTiratore` il costruttore è il costruttore di default, e il suo effetto è di chiamare il costruttore di default della superclasse
[4] qui c'è un problema di tipi: `xy` dovrebbe essere di un supertipo e non di un sottotipo del valore che le viene assegnato

Domanda 6 L'istruzione `Politico xy = new Onorevole();`
☐ dà errore in esecuzione perchè la classe `Onorevole` non ha il costruttore;
☐ dà errore in compilazione perchè l'assegnazione viola il sistema dei tipi;
☑ dà errore in compilazione perchè la classe `Onorevole` è astratta.

Domanda 7 L'istruzione `Politico franceschini = new OnorevoleOpposizione();`
☐ dà errore in esecuzione;
☑ dà errore in compilazione;[5]
☐ stamperà `Onorevole silenzioso creato`.

Domanda 8 L'istruzione `Legge centotrentatre = new LeggeTaglio("D. L. 133");`
☑ stamperà `D.L. 133`[6]
`sara' approvata`
`con lacrime e sangue`;
☐ stamperà `D.L. 133`;
☐ dà errore in compilazione.

Domanda 9 L'istruzione `LeggeTaglio centottanta = new LeggeTaglio("D. L. 180");`
☑ stamperà `D.L. 180`
`sara' approvata`
`con lacrime e sangue`;
☐ stamperà `D.L. 180`;
☐ dà errore in compilazione.

Domanda 10 L'istruzione `alfano.vota(centotrentatre);` stamperà:
☐ `SI`
☑ `NO`[7]
☐ dà errore in compilazione perchè il metodo `vota` non è definito in `OnorevoleMaggioranza`;
☐ dà errore in esecuzione, perchè il tipo dinamico di `centotrentatre` è `LeggeTaglio` e non `Legge`.

Domanda 11 L'istruzione `alfano.vota(centottanta);` stamperà:
☐ `SI`
☑ `NO`
☐ dà errore in compilazione perchè il metodo `vota` non è definito con parametro di tipo `LeggeTaglio`;
☐ dà errore in esecuzione.

[5] Il costruttore con un parametro definito in `OnorevoleOpposizione` cancella il costruttore di default.

[6] in questo caso, come alla domanda seguente, ogni costruttore chiama il costruttore della superclasse

[7] osservate (in questa e nella prossima domanda, che la scelta del metodo `appoggia` da chiamare avviene all'interno del codice del metodo `vota`, dove il parametro `l` ha *sempre* tipo statico `Legge`.

Domanda 12 L'istruzione `if (alfano.appoggia(centotrentatre)) P.print("SI"); else P.print("NO");` stamperà:
☐ `SI`
☑ `NO`
☐ dà errore in esecuzione.

Domanda 13 L'istruzione `if (alfano.appoggia((LeggeTaglio) centotrentatre)) P.print("SI"); else P.print("NO");` stamperà:
☑ `SI`[8]
☐ `NO`
☐ dà errore in compilazione;
☐ dà errore in esecuzione.

Domanda 14 L'istruzione `if (alfano.appoggia(centottanta)) P.print("SI"); else P.print("NO");` stamperà:
☑ `SI`
☐ `NO`
☐ dà errore in esecuzione.

Domanda 15 L'istruzione `diPietro.vota(centotrentatre);` stamperà:
☐ `SI`
☑ `NO`
☐ dà errore in compilazione perchè il metodo `vota` non è definito in `OnorevoleMaggioranza`;
☐ dà errore in esecuzione perchè il tipo dinamico di `centotrentatre` è `LeggeTaglio` e non `Legge`;

Domanda 16 L'istruzione `diPietro.vota(centottanta);` stamperà:
☐ `SI`
☑ `NO`[9]
☐ dà errore in compilazione perchè il tipo statico di `centottanta` è `LeggeTaglio` e non `Legge`;

Domanda 17 L'istruzione `if (diPietro.appoggia(centotrentatre)) P.print("SI"); else P.print("NO");` stamperà:
☐ `SI`
☑ `NO`
☐ dà errore in compilazione;
☐ dà errore in esecuzione.

[8] al solito l'overloading viene risolta staticamente sulla base del tipo statico dei parametri attuali: il downcast modifica il tipo statico!

[9] In questa domanda e nella precedente, occorre osservare che il comportamento del metodo `appoggia` nella classe `OnorevoleOpposizione` dipende dal *tipo dinamico* del parametro e non è influenzato quindi dai tipi statici.

Domanda 18 L'istruzione `if (diPietro.appoggia((Legge) centottanta)) P.print("SI"); else P.print("NO");` stamperà:
☐ `SI`
☑ `NO`
☐ dà errore in compilazione perchè `centottanta` è sottotipo di `Legge`;
☐ dà errore in esecuzione, perchè il tipo dinamico di `centottanta` non è sottotipo di `Legge`.

Domanda 19 L'istruzione `if (diPietro.appoggia((LeggeOpposizione) centotrentatre)) P.print("SI"); else P.print("NO");` stamperà:
☐ `SI`
☐ `NO`
☐ dà errore in compilazione perchè `Legge` non è sottotipo di `LeggeOpposizione`;
☑ dà errore in esecuzione, perchè il tipo dinamico di `centotrentatre` non è sottotipo di `LeggeOpposizione`[10].

Domanda 20 L'istruzione `if (centotrentatre.costo()>0) P.print("buona legge"); else P.print("accidenti!");` stamperà:
☐ `buona legge`
☐ `accidenti!`
☑ dà errore in compilazione, perchè il metodo `costo()` non è definito sulla classe `Legge`;
☐ dà errore in esecuzione.

Domanda 21 L'istruzione `if (salvaBanche.costo()>0) P.print("buona legge"); else P.print("accidenti!");` stamperà:
☐ `buona legge`
☑ `accidenti!`[11]
☐ dà errore in compilazione;
☐ dà errore in esecuzione.

[10] qui occorre ricordare che Java controlla run-time la correttezza dei downcast: in questo caso, l'oggetto ha tipo dinamico che non è sottotipo di `LeggeOpposizione`, per cui verrà sollevata l'eccezione `ClassCastException`.

[11] osservate che il codice chiamato dipende dal tipo dinamico di `salvaBanche`, che è `LeggeTaglio`, mentre staticamente, a differenza della domanda precedente, il tipo statico di `salvaBanche`, cioè `LeggeGoverno` contiene nella sua interfaccia il metodo `costo()`.

Domanda 22 L'istruzione `if (mastella.appoggia(centotrentatre)) P.print("SI"); else P.print("NO");` stamperà:
☐ `SI`
☑ `NO`[12]
☐ dà errore in compilazione;
☐ dà errore in esecuzione.

Domanda 23 L'istruzione `if (mastella.appoggia(centottanta)) P.print("SI"); else P.print("NO");` stamperà:
☐ `SI`
☑ `NO`[13]
☐ dà errore in compilazione;
☐ dà errore in esecuzione.

Domanda 24 L'istruzione `if (mastella.appoggia(salvaBanche)) P.print("SI"); else P.print("NO");` stamperà:
☑ `SI`[14]
☐ `NO`
☐ dà errore in compilazione;
☐ dà errore in esecuzione.

Domanda 25 L'istruzione `Legge l = gelmini.propone("Nuovi Tagli");` causerà:
☐ Un errore in compilazione, perchè il metodo `propone` restituisce un oggetto di tipo `LeggeTaglio`;
☐ Un errore in esecuzione, perchè il metodo `propone` restituisce un oggetto di tipo dinamico `LeggeTaglio`;
☐ un errore in compilazione perchè il metodo `propone` non è definito nell'interfaccia del tipo statico di `gelmini`;
☑ compila ed esegue correttamente[15].

Domanda 26 L'istruzione `gelmini.esclama();` causerà:
☐ Un errore in compilazione, perchè il metodo `esclama()` non è definito nella classe `Ministro`;
☐ Un errore in esecuzione, perchè la variabile `fraseTipica` non è stata inizializzata;
☐ stamperà la stringa vuota;
☑ stamperà `null`[16];

[12] infatti `centrotrentatre` staticamente è una `Legge`, per cui si applica il metodo `appoggia(Legge)` della classe `OnorevoleMaggioranza`

[13] Qui viene scelto il metodo `appoggia(LeggeTaglio)` che inverte il valore di verità del metodo `appoggia(LeggeGoverno)` chiamato attraverso `super` (ricordate la regola del *best match*).

[14] sempre a causa del fatto che Java considera diversi metodi che hanno tipi diversi, la definizione di `appoggia` nella classe `FrancoTiratore` non fa overriding dei metodi `appoggia` nella classe `OnorevoleMaggioranza`. Qui il best match è col metodo `appoggia(LeggeGoverno)`.

[15] questo, ovviamente non necessita di spiegazioni `:-)`

[16] La creazione di un Ministro con il costruttore senza parametri inizializza la variabile

Domanda 27 L'istruzione `Legge l = brunetta.propone("Nuovi Tagli");` causerà:
☐ Un errore in compilazione, perchè il metodo `propone(String)` restituisce un oggetto di tipo `LeggeTaglio`
☐ Un errore in esecuzione, perchè il metodo `propone(String)` restituisce un oggetto di tipo dinamico `LeggeTaglio`
☑ un errore in compilazione perchè il metodo `propone(String)` non è definito nell'interfaccia del tipo statico di `brunetta`;
☐ compila ed esegue correttamente;

Domanda 28 L'istruzione `veltroni.esclama();` causerà:
☑ Un errore in compilazione, perchè il metodo `esclama()` non è definito nella classe `Politico`;
☐ Un errore in esecuzione, perchè l'oggetto `veltroni` non ha tipo dinamico `Onorevole`;
☐ stamperà `null`;
☐ stamperà `Siamo per la pace, ma anche per la giustizia`;

Domanda 29 L'istruzione `((Onorevole) veltroni).esclama();` causerà:
☐ Un errore in compilazione, perchè il metodo `esclama()` non è definito nell'interfaccia `Politico`;
☐ Un errore in esecuzione, perchè l'oggetto `veltroni` non ha tipo dinamico `Onorevole`;
☐ stamperà `null`;
☑ stamperà `Siamo per la pace, ma anche per la giustizia`;

Domanda 30 L'istruzione `LeggeTaglio l = ((Ministro) brunetta).propone("Chiudo le Poste");` causerà:
☐ Un errore in compilazione, perchè il metodo `propone(String)` non è definito nell'interfaccia `Politico`;
☐ Un errore in esecuzione, perchè il downcast è illegale;
☑ compila ed esegue correttamente.

`fraseTipica` al valore di default del tipo `String`. Il valore di default per tutti i tipi che sono classi è `null`.