Esonero del corso di Programmazione a Oggetti

Roma, 1 dicembre 2005

Considerate le seguenti definizioni di classi e interfacce in Java:

```
interface Fumetto{ void esclama();
                   void utile();}

class Personaggio implements Fumetto{
   private String nome = "";
   public void esclama(){print("gulp");}
   public void utile(){if (salvaIlMondo()) print("personaggio utile");
                          else print("personaggio inutile");}
   protected boolean salvaIlMondo(){return false;}
   protected void print(String s) {System.out.println(s);}
   public Personaggio() {print("sono nato");}
   public Personaggio(String s) {nome=s;}
}

class Papero extends Personaggio{
   public void esclama() {print ("quack");}
   public void pubblicatoDa() {print("Disney");}
   public Papero() {print("si e' schiuso l'uovo");}
   public Papero(String s) {super(s);}
}

class SuperEroe extends Personaggio{
   public void esclama() {print("accidenti");}
   protected boolean salvaIlMondo(){return true;}
   public SuperEroe() {print("eccomi!");}
   public SuperEroe(String s){super(s);}
}

class Xman extends SuperEroe{
   public void esclama() {print("per tutti i mutanti!");}
   public Xman() {print("sono mutato!");}
   public void pubblicatoDa(){print("Marvel Comics");}
   public void esclama(int x){print(""+x+"volte sono mutato");}
}
```

Domanda 1 Quali delle seguenti dichiarazioni produce un errore di tipo in compilazione:
☑ `Xman wolverine = new SuperEroe();`
☐ `SuperEroe wolverine = new Xman();`
☐ `Xman wolverine = new Xman();`
▷ Infatti `SuperEroe` non é sottotipo di `Xman`

Domanda 2 Quali delle seguenti dichiarazioni produce un errore in compilazione:
☐ `SuperEroe wolverine = new SuperEroe("Wolverine");`
☑ `SuperEroe wolverine = new Xman("Wolverine");`
☐ `Personaggio wolverine = new SuperEroe("Wolverine");`
▷ Infatti la classe `Xman` non ha il costruttore con un parametro

Domanda 3 Considerate la dichiarazione: `Fumetto f = new Fumetto();`
☐ E' corretta;
☐ E' scorretta perché una variabile non può essere dichiarata di tipo interfaccia;
☑ E' scorretta perché non si possono istanziare oggetti dalle interfacce.

Domanda 4 La creazione di oggetto `Fumetto gastone = new Papero("gastone");` produce in output:
☐ `sono nato`
`si e' schiuso l'uovo`
☐ `si e' schiuso l'uovo`
☑ nessun output
▷ Infatti il costruttore con parametro della classe `Papero` si limita a settare la variabile `nome`

Domanda 5 Considerate l'istruzione `Fumetto mystica = new Xman();` Essa produrrà l'output:
☑ `sono nato`
`eccomi!`
`sono mutato!`
☐ `sono mutato!`
`eccomi!`
`sono nato`
☐ `sono mutato!`
▷ Infatti i costruttori delle sottoclassi prima chiamano i costruttori delle superclassi.

Domanda 6 Data la dichiarazione: `Fumetto magneto = new Xman();` l'invocazione del metodo `magneto.pubblicatoDa();`
☐ produce l'output `Marvel Comics`
☐ produce un errore in esecuzione, perché il l'oggetto `magneto` non può rispondere al metodo `pubblicatoDa()`
☑ produce un errore in compilazione, perché il il tipo `Fumetto` non prevede nella sua interfaccia il metodo `pubblicatoDa()`

Domanda 7 Data la dichiarazione: `Personaggio tempesta = new Xman();` l'invocazione del metodo `tempesta.esclama();`
☐ produce l'output `gulp`
☑ produce l'output `per tutti i mutanti!`
☐ produce un errore in esecuzione, perché non è possibile determinare il codice da eseguire
▷ Infatti la scelta del metodo dipende dal tipo dinamico. Osservare che tra i due metodi `esclama` della classe `Xman` si sceglie quello senza parametri (*overloading*).

Domanda 8 Data la dichiarazione `Personaggio colosso = new Xman();` l'invocazione del metodo `colosso.utile();`
☑ produce l'output `personaggio utile`
☐ produce l'output `personaggio inutile`
☐ produce un errore in esecuzione, perché il metodo `utile` non è definito nella classe `Xman`
▷ Attenzione al *dynamic method lookup*: `salvaIlMondo` é ridefinito nella classe `SuperEroe`! E la ricerca del metodo ricomincia sempre dalla classe a cui appartiene l'oggetto ricevente!

Domanda 9 Data la dichiarazione `Xman colosso = new Xman();` l'invocazione del metodo `colosso.utile();`
☑ produce l'output `personaggio utile`
☐ produce l'output `personaggio inutile`
☐ produce un errore in esecuzione, perché il metodo `salvaIlMondo()` non é definito nella classe `Xman`
▷ Come sopra: non é rilevante che cambi il tipo statico della variabile `colosso`.

Domanda 10 Data la dichiarazione `Personaggio paperinik = new Papero();` l'invocazione del metodo `paperinik.utile();`
☐ produce l'output `personaggio utile`
☑ produce l'output `personaggio inutile`
☐ produce un errore in esecuzione, perché il metodo `utile()` non é definito nella classe `Papero`
▷ Beh, non fatevi confondere dal fatto che `paperinik` si atteggia a `SupeEroe :-)`

Domanda 11 Data la dichiarazione `Personaggio paperone = new Papero();` l'invocazione del metodo `paperone.pubblicatoDa();`
☐ produce l'output `Disney`
☐ produce l'output `Marvel Comics`
☑ produce un errore in compilazione, perché il metodo `pubblicatoDa()` non appartiene all'interfaccia del tipo `Personaggio`
▷ Il compilatore controlla i *tipi statici* e poco importa al compilatore se al run-time `paperone` sarà un `Papero`, invece di un `Personaggio`.

Domanda 12 Data la dichiarazione `Papero paperino = new Papero();` l'invocazione del metodo `paperino.pubblicatoDa();`
☑ produce l'output `Disney`
☐ produce l'output `Marvel Comics`
☐ produce un errore in compilazione, perché il metodo `pubblicatoDa()` non appartiene all'interfaccia `Fumetto`

Domanda 13 Data la dichiarazione `SuperEroe spiderman = new SuperEroe();` l'invocazione del metodo `spiderman.pubblicatoDa();`
☐ produce l'output `Disney`
☐ produce l'output `Marvel Comics`
☑ produce un errore in compilazione, perché il metodo `pubblicatoDa()` non è definito nella classe `SuperEroe`

Domanda 14 Considerare la definizione di classe:

```
class Antenato implements Fumetto{
 public void esclama(System.Out.println("yabadabadu"))}
}
```

☐ E' corretta;
☑ E' scorretta perché non implementa il metodo `utile();`
☐ E' scorretta perché non definisce il metodo costruttore;
▷ Una classe, se estende un'interfaccia, deve dichiarare *pubblici* tutti i metodi dichiarati nell'interfaccia.

Domanda 15 Considerare la definizione di classe:

```
class Antenato extends Personaggio{
 public void esclama(){System.Out.println("yabadabadu");}
}
```

☑ E' corretta;
☐ E' scorretta perché non implementa il metodo `salvaIlMondo();`
☐ E' scorretta perché non definisce il metodo costruttore.
▷ Osservare che sarebbe corretta anche `class Antenato extends Personaggio implements Fumetto`, perché il metodo `utile()` si erediterebbe da `Personaggio`

Domanda 16 Relativamente alle definizione date, quale delle seguenti affermazioni è vera:
☐ `Papero` è sottotipo di `SuperEroe`;
☐ `SuperEroe` è sottotipo di `Papero`:
☑ I tipi `Papero` e `SuperEroe` non sono in nessuna relazione rispetto alla nozione di sottotipo.

Domanda 17 Relativamente alle definizione date, quale delle seguenti affermazioni è vera:
☑ `Personaggio` è sottotipo di `Fumetto`
☐ `Fumetto` è sottotipo di `Personaggio`
☐ `SuperEroe` è sottotipo di `Papero`

Domanda 18 Relativamente alle definizione date, quale delle seguenti affermazioni è vera:
☑ Tutti i tipi definiti sono sottotipi di `Fumetto`;
☐ Solo il tipo `Personaggio` è sottotipo di `Fumetto`;
☐ Nessun tipo definito è sottotipo di `Fumetto`.

Domanda 19 Considerate una classe `TestaFumetti` in cui sia dichiarato il seguente metodo:

```
public static void esclama(Fumetto f){f.esclama()}
```

Considerate la dichiarazione `Personaggio dorettaDoremi = new Papero();`. L'invocazione del metodo: `TestaFumetti.esclama(dorettaDoremi)`
☐ E' scorretta perché `dorettaDoremi` non è di tipo `Fumetto`;
☐ E' scorretta perché `TestaFumetti` non è un oggetto;
☑ produce l'output `quack`.
▷ Ricordare che i *metodi statici* si invocano sulle classi e i parametri formali possono essere supertipi dei parametri attuali.

Domanda 20 Considerate la definizione: `Fumetto[] F;` L'array `F` potrà contenere:
☐ Solo oggetti di tipo `Fumetto`;
☐ Solo oggetti istanziati con una operazione della forma `new Fumetto();`
☑ Solo oggetti che hanno un tipo dinamico sottotipo di `Fumetto`,
Domanda 21 Considerate una classe `TestaFumetti` in cui sia dichiarato il seguente metodo:

```
public static void esclama(Personaggio[] f)
   {for (int i=0; i<f.length; i++) f[i].utile();}
```

☐ Il metodo darà errore in esecuzione se un qualche elemento dell'array fosse di tipo `Xman`;
☐ Il metodo stamperà solo la stringa `personaggio inutile` un numero di volte pari alla lunghezza del vettore;
☑ Il metodo stamperà un po' di volte la stringa `personaggio inutile` e un po' di volte la stringa `personaggio utile` a seconda del tipo dinamico degli elementi presenti nell'array.
▷ infatti il lookup del metodo dipenderà dal tipo dinamico, mentre staticamente si é controllato che tutti gli elementi dell'array siano sottotipo di `Personaggio`.
Domanda 22 Considerate una classe `TestaFumetti` in cui sia dichiarato il seguente metodo:

```
public static void esclama(Personaggio[] f)
   {for (int i=0; i<f.length; i++) f[i].pubblicatoDa();}
```

☑ Il metodo darà errore in compilazione perché il metodo `pubblicatoDa()` non è definito sul tipo `Personaggio`;
☐ Il metodo stamperà un po' di volte la stringa `Disney` e un po' di volte la stringa `Marvel Comics` a seconda del tipo dinamico degli elementi presenti nell'array;
☐ Il metodo potrebbe dare errori in esecuzione se il tipo dinamico di un elemento fosse `Personaggio`, mentre funzionerebbe se tutti gli elementi dell'array avessero tipo dinamico `Papero` o `Xman`.
Domanda 23 Considerata la dichiarazione: `Fumetto uomoGhiaccio = new Xman();`.
L'invocazione di metodo `uomoGhiaccio.esclama(3);`
☐ produce in output `3 volte sono mutato`;
☐ produce in output `sono mutato!`;
☑ produce un errore in compilazione.
▷ perché staticamente `uomoGhiaccio` ha tipo `Fumetto`, che non definisce il metodo `esclama(int)`
Domanda 24 Considerata la dichiarazione: `SuperEroe paperinik = new Papero();`.
☐ compila ed esegue correttamente;
☐ produce un errore in esecuzione;
☑ produce un errore in compilazione.
▷ vedi risposta domanda 16.