Esercizi d'esame per Fondamenti A II

1) Domande a risposta multipla (radiobutton)

Si definisca una classe A che implementa le interfacce I1, I2 e I3. Valga inoltre:
```
interface I3 extends I1;
```
Quale delle seguenti affermazioni è corretta:
1. A deve implementare tutti i metodi delle tre interfacce.
2. A può implementare tutti i metodi delle tre interfacce.
3. A deve implementare tutti i metodi delle sole interfacce I1 e I2.
4. Errore di compilazione: l'interfaccia I3 non può estendere un'altra interfaccia.

Un frammento di codice usa le classi ClasseA, ClasseB (che eredita da ClasseA), e la ClasseC (che eredita da ClasseB). Tutte le classi implementano la interfaccia IntI.
```
ClasseA a;  ClasseB b; ClasseC c; IntI i;
```
1. Il polimorfismo verticale consente di assegnare a = i; b = i; c = i;
2. Il polimorfismo orizzontale consente di assegnare i = c; i = a; a = i; c = i;
3. Il polimorfismo verticale consente di assegnare a = b; a = c; b= c;
4. Il polimorfismo orizzontale consente di assegnare a = i; b = i; c = i;

Sono state definite in questo ordine
una classe A astratta che ha alcuni metodi implementati e che implementa la interfaccia IA;
una interfaccia IA che specifica l'insieme di metodi della classe A
Quale affermazione è vera?
1. Non sono rispettati i vincoli temporali di sviluppo.
2. Non è possibile che una classe astratta A abbia alcuni metodi implementati.
3. Non è possibile che una classe A abbia una interfaccia che coincide con quella di una interfaccia.
4. Si possono creare istanze dalla classe A.

Sono definite
una classe A che definisce i metodi ma, mb e mc;
una sottoclasse B che eredita da A e che definisce i metodi ma, mb e mc.
Quale affermazione è vera per le classi?
1. Il metodo ma della classe A e il metodo ma della classe B sono in overriding se i metodi hanno gli stessi parametri di ingresso e uscita, e le stesse eccezioni.
2. Il metodo mb della classe A e il metodo mb della classe B sono in overriding se i metodi hanno gli stessi parametri di ingresso e uscita.
3. I metodi con lo stesso nome della classe A e della classe B sono in overloading.
4. Non si possono definire classi che abbiano i metodi completamente ricoperti (in overriding).

Indicare la risposta corretta per l'ambiente Java
1. L'uso del compilatore e dell'interprete del bytecode consente la massima portabilità tra architetture diverse.
2. L'uso dell'interprete consente la massima efficienza anche per architetture diverse.

3. La esecuzione del bytecode consente di evitare errori che un compilatore non avrebbe potuto evitare.
4. Non si possono realizzare compilatori per l'ambiente Java.

Come si possono mettere in relazione le visibilità ottenibili dalle clausole: package, protected
1. protected è più restrittiva di package.
2. package è più restrittiva di protected.
3. Non si possono mettere in relazione.
4. package è solo in relazione a private.

In un frammento di codice si usano le classi ClasseA, ClasseB (che eredita da ClasseA), e ClasseC (che eredita da ClasseB)

```
ClasseA a;  ClasseB b1, b2; ClasseC c;
b1 = new ClasseA ();  // A
b2 = new ClasseB ();  // B
c = b2;               // C
a = b1;               // D
```

1. L'assegnamento B è errato.
2. L'assegnamento C è errato
3. Gli assegnamenti A e C sono errati.
4. L'assegnamento D è errato.

Un frammento di codice usa le classi ClasseA, ClasseB (che eredita da ClasseA), e ClasseC (che eredita da ClasseB). Tutte le classi implementano la interfaccia IntI.

```
ClasseA a;  ClasseB b1, b2; ClasseC c; IntI i;
```

1. Il polimorfismo verticale consente di assegnare b1 = c;
2. Il polimorfismo orizzontale consente di assegnare b1 = c;
3. Il polimorfismo verticale consente di assegnare c = b1;
4. Il polimorfismo orizzontale consente di assegnare b1 = i;

Le linee di codice seguenti all'interno di un file sorgente che definisce la classe Grafica:

```
import java.awt.event.*;
```

1. aggiungono codice sorgente alla classe Grafica.
2. è indispensabile se la classe Grafica vuole disegnare componenti e gestire interazione con l'utente.
3. è indispensabile se la classe Grafica vuole disegnare componenti .
4. consente di usare il nome ActionEvent anziché ricorrere al nome completo java.awt.event.ActionEvent.

Sia b un oggetto di classe JButton e o1, o2 due oggetti qualsiasi. Dato il codice:
b.addActionListener(o1);
b.addActionListener(o2);
quale affermazione risulta vera?
1. Gli oggetti fungono entrambi da Listener degli eventi del bottone e i metodi di grafica sono invocati a turno per gestire il bottone b.

2. Vengono invocati i metodi actionPerformed() degli oggetti o1, o2, se almeno una delle due classi implementa la interfaccia ActionListener.
3. Vengono invocati i metodi actionPerformed() di entrambi gli oggetti o1, o2, se le classi relative implementano la interfaccia ActionListener.
4. Viene invocato il metodo actionPerformed() di quello dei due oggetti o1, o2, la cui classe è più vicina alla classe Graphics.

Il frammento di codice:
MyClass[] o = new MyClass[4];
for (int i=0; i < 4; i++) {o[i]= new MyClass (i);}

1. produce un oggetto o, array di 4 elementi, ciascuno dei quali è un oggetto MyClass inizializzato con un costruttore con parametro la posizione.
2. produce un oggetto o, array di 4 elementi, ciascuno dei quali è un riferimento ad un oggetto MyClass inizializzato con un costruttore con parametro la posizione.
3. produce un oggetto o, array di 4 elementi, ciascuno dei quali è un oggetto MyClass inizializzato a null.
4. crea una nuova lista di quattro elementi di tipo MyClass.

Si consideri un main, invocato con argomenti di invocazione: a1 12 “pippo pluto”

```
public class MainClass {
 public static void main(String[] args){
  int as = 0;
  for (int i=0; i < args.length; i++)
   {System.out.println (args[i]); as+= i;}
 System.out.println (as);
}}
```

1. Il processo di esecuzione stampa i tre argomenti di invocazione e 3.
2. Il processo di esecuzione stampa i quattro argomenti di invocazione e 4.
3. Il processo di esecuzione stampa i tre argomenti di invocazione e 6.
4. Il processo di esecuzione stampa i tre argomenti di invocazione e null.

Che effetto produce il seguente blocco di codice se la stringa s contiene “pippo”?

```
try {a = Double.parseDouble(s);}
catch (NumberFormatException e) {
 System.out.println("Stringa mal fatta");
}
```

1. Il programma produce in a il valore 5.4, che deriva dal parsing di s.
2. Il programma produce una eccezione, stampa un messaggio di errore e termina.
3. Il programma produce una eccezione, stampa un messaggio di errore e continua.
4. Il programma produce una eccezione e termina.

Se un metodo ha la seguente definizione:

```
public Object pop() throws StackEmptyException;
```

cosa vale tra i seguenti:

1. Il metodo pop restituisce un oggetto di tipo numerico o una eccezione.
2. Il metodo pop restituisce un oggetto di una qualunque classe o una eccezione.

3. Il metodo pop restituisce un oggetto di un tipo numerico inclusi i primitivi.
4. Il metodo pop restituisce un oggetto di qualunque classe inclusi i primitivi o una eccezione.

Se in un metodo compaiono le seguenti istruzioni:

```
if (x>100000) throw new NumberTooBigException();
```

quale delle seguenti affermazioni è falsa?

1. Il metodo deve prevedere un clausola throws NumberTooBigException.
2. Il metodo può restituire o l'oggetto del tipo previsto o una eccezione.
3. Il metodo restituisce sempre un oggetto del tipo previsto.
4. Il programma intero potrebbe terminare a causa delle istruzioni riportate sopra.