Se una variabile di istanza x viene dichiarata private all’interno di una classe Prova, allora:

Solo i metodi appartenenti alla classe Prova possono accedere a x

**abstract** **class** E{

**abstract** **int** f();

**abstract** **int** g();

**int** h(){**return** 0;}

}

Quali tra le seguenti classi sono definite correttamente? NESSUNA

**class** F1 **extends** E{

**int** f() {**return** 1;}

**int** h() {**return** 1;}

}

**class** F2 **extends** E{

**int** f(**int** x) {**return** x;}

**int** g(**int** y) {**return** y;}

**int** h(**int** z) {**return** z;}

}

**class** F3 **extends** E{

**abstract** **int** f();

**abstract** **int** g();

**int** h {**return** 3;}

}

Si considerino le classi A e Prova riportate di seguito:

**class** A{

**private** **int** n;

**public** A(**int** n) {**this**.n=n;}

**public** **boolean** equals(Ax){

**return** **this**.n == x.n;

}

}

**class** Prova{

**public** **static** **void** main(String[] args){

Object x = **new** A(10);

Object y = **new** A(10);

System.out.println(x.equals(y));

System.out.println(((A)x).equals(((A)y));

}

}

Il metodo main della classe Prova stampa prima **false** e poi **true**

**import** java.util.\*

**class** Test{

**public** **static** **void**...{

List<String> v = **new** ArrayList<String>();

v.add(0,"A");

v.add(0,"B");

v.add(0,"C");

Iterator<String>i;

**for**(i=v.iterator();i.hasNext();i.next()){

System.***out***.println(i.next());

}

}

}

La classe test genera un errore a tempo di esecuzione

Riguardo la specifica dei tipi astratti di dato, una sola delle seguenti affermazioni è vera: Un tipo astratto di dato è specificato mediante un elenco di domini e un elenco di funzioni che operano su tali domini.

Si considerino le classi Solido e ProvaSolido riportate di seguito:

public abstract class Solido {

public double volume;

}

public class ProvaSolido {

public static void main (String[] args) {

Solido s=new Solido ();

System.out.println (s.volume);

}

}

La classe ProvaSolido causa un errore a tempo di compilazione, perché non è possibile istanziare un oggetto appartenente ad una classe astratta.

public class Punto {

private int x, y;

public Punto(int x, int y) { this.x = x; this.y = y; }

public boolean equals(Object p){

if (p==null || !p.getClass().equals(getClass())) return false;

return this.x == ((Punto)p).x && this.y == ((Punto)p).y;

}

public static void main(String[] args) {

Punto p1 = new Punto(0,0);

Punto p2 = new Punto(0,0);

Object o2 = p2;

System.out.println(p1.equals(p2));

System.out.println(p1.equals(o2));

}

}

La classe Punto, una volta compilata ed eseguita, stampa due volte true.

public static void cosaFa(int[] v) {

int temp;

for (int i=0; i<(v.length/2); i++) {

temp = v[i];

v[i] = v[v.length - 1 - i];

v[v.length - 1 - i] = temp;

}

}

Qual è il risultato dell’invocazione del metodo cosaFa su un array di interi? Viene modificato l’array passato come argomento rovesciando l’ordine degli elementi.

Fare overloading significa dare diverse definizioni di metodi con lo stesso nome, ma usando parametri formali diversi in numero e/o tipo.

Nel corpo di un metodo, “this” contiene sempre il riferimento all’oggetto su cui quel metodo è invocato.

double y = 13.0;  
int x = y;

L’assegnazione int x = y è sbagliata, ma può essere corretta rimpiazzando y con (int) y🡪 int x = (int) y

Il break può essere usato per interrompere un ciclo, ma è consentito scrivere cicli senza usare break.

Durante l’invocazione di un metodo, i parametri attuali vengono copiati nei corrispondenti parametri formali.

public in mioMetodo(int y){

x = y+1;

return x

}

Il metodo è sbagliato in ogni caso, poiché x non è dichiarato in mioMetodo.

Si considerino le seguenti espressioni:

"Fondamenti".concat(" di").concat(" Informatica")

"Fondamenti".concat(" di".concat(" Informatica"))

Entrambe le espressioni restituiscono la stringa “Fondamenti di Informatica”

Quali delle seguenti affermazioni relative al ciclo while è errata?

La condizione del ciclo while viene valutata solo una volta.

Quale delle seguenti affermazioni relative al ciclo while è vera?

Il corpo di un ciclo while potrebbe non essere mai eseguito.

public static int[] mioMetodo(int[] s) {

int[] v = new int[s.length];

for (int i=0 ; i < s.length; i++)

v[i] = s[i+1];

return v;

}

Il metodo genera un errore di esecuzione di tipoArrayIndexOutOfBoundsException sull'istruzione v[i] = s[i+1], poichè l'indice i+1 non è un indice valido.

Siano a,b,c tre variabili di tipo int. L'espressione di tipo boolean !((a!=b) && (a!=c))restituisce valore vero se:

a ha un valore uguale a b oppure uguale a c.

Quale delle seguenti affermazioni relative a == è errata?

== restituisce sempre falso se i suoi argomenti non sono di tipo primitivo.

Le variabili di istanza sono visibili: In tutti i metodi non statici della classe a meno che non vengano mascherate da dichiarazioni locali.

public static double f(double x, int y){

double z = 1.0;

int k;

if (y<0) k=-y;

else k=y;

for (int i=0; i<k; i++)

z \*= x;

if (y<0) return 1/z;

else return z;

}

Il metodo calcola x elevato alla y.

Si consideri la seguente espressione Java: !a && b || c. Una sola delle seguenti affermazioni è vera. Quale?  
L'espressione è equivalente a: ((!a) && b) || c

public static int f(int[] x, int y){

int c=0;

for (int i=0; i<x.length; i++)

if (x[i]>y) c++;

return c;

}

Il metodo restituisce il numero di elementi di x più grandi di y.

Una volta creato un array, non è possibile modificarne la dimensione.

a==b, dove a e b sono riferimenti ad oggetti di tipo String.

L'espressione vale true se a e b contengono lo stesso indirizzo, cioè puntano al medesimo oggetto String.

public static void add(int y){

this.x=y;

}

Il metodo è errato, perchè non è possibile usare this in un metodo statico.