

Cuestionario 3;

Monday, November 25, 2024 5:19 PM

1. Which declaration initializes a boolean variable?
 - a) boolean m = null
 - b) Boolean j = (1<5)
 - c) boolean k = 0
 - d) boolean h = 1

Un boolean solo puede ser true o false.

//

2. What is the DTO pattern used for?

- a) To Exchange data between processes
- b) To implement the data Access layer
- c) To implement the presentation layer

¿Qué es el patrón DTO?

El patrón **Data Transfer Object (DTO)** se utiliza para encapsular datos y transferirlos entre procesos, capas de una aplicación o incluso entre sistemas distribuidos. Objetivo principal;

- **Optimizar la transferencia de datos:** Reducir las llamadas remotas o el volumen de datos que viajan en la red al empaquetar datos relevantes en un solo objeto.
- **Separación de responsabilidades:** El DTO no contiene lógica de negocio, solo datos simples que necesitan ser enviados o recibidos.
- **Estandarización:** Sirve como una interfaz clara para compartir información, evitando la exposición directa de los modelos de dominio.

///

3. What value should replace kk in line 18 to cause jj = 5 to be output?

```
public class MyFive {  
    public static void main(String[] args) {  
        //short kk = ?;  
        short ii;  
        short jj = 0;  
        for (ii = kk; ii > 6; ii--) {  
            jj++;  
        }  
        System.out.println("jj = " + jj);  
    }  
}
```

- a) -1
- b) 1
- c) 5
- d) 8
- e) 11

```

1 MyFive.java X
2 public class MyFive {
3     public static void main(String[] args) {
4         short kk = 11;
5         short ii;
6         short jj = 0;
7         // 11           ii = ii - 1
8         for (ii = kk; ii > 6; ii-=1) {
9             jj++; //0 (ENTRAR 5 VECES)
10        }
11    }
12
13    System.out.println("jj = " + jj); //5
14
15    // kk  ii  jj
16    // 11  11  0
17    //          1
18    //          10 2
19    //          9  3
20    //          8  4
21    //          7  5
22    //          6
23 }

```

///

4. ¿Cuál será el resultado?

```

public class SampleClass {
    public static void main(String[] args) {
        SampleClass sc, scA, scB;
        sc = new SampleClass();
        scA = new SampleClassA();
        scB = new SampleClassB();
        System.out.println("Hash is: " + sc.getHash() +
                           ", " + scA.getHash() + ", " + scB.getHash());
    }
    public int getHash() {
        return 111111;
    }
}
class SampleClassA extends SampleClass {
    public int getHash() {
        return 44444444;
    }
}
class SampleClassB extends SampleClass {
    public int getHash() {
        return 999999999;
    }
}

```

- a) Compilation fails
- b) An exception is thrown at runtime
- c) There is no result because this is not correct way to determine the hash code
- d) Hash is: 111111, 44444444, 99999999.

El método `getHash()` invocado depende del tipo real del objeto, no del tipo de referencia.

Las clases hijas redefinen el comportamiento de `getHash()` para devolver valores diferentes (Override)

///

5. ¿Cuál sería el resultado?

```
public class DoCompare4 {  
    public static void main(String[] args) {  
        String[] table = {"aa", "bb", "cc"};  
        int ii = 0;  
        do {  
            while (ii < table.length) {  
                System.out.println(ii++);  
            }  
        } while (ii < table.length);  
    }  
}
```

Al ser un do while, hace una primera instancia y despues evalua el while, la longitud de la cadena de Strings es 3, es por eso que solo imprimiría

0 1 2

///

6. ¿Cuál sería el resultado?

```
public class DoCompare1 {  
    public static void main(String[] args) {  
        String[] table = {"aa", "bb", "cc"};  
        for (String ss : table) {  
            int ii = 0;  
            while (ii < table.length) {  
                System.out.println(ss + ", " + ii);  
                ii++;  
            }  
        }  
    }  
}
```



Ciclo for each tiene anidado un while, el for each hará que se recorra sobre todos los elementos del arreglo table. Así que imprimira cada posible combinacion mientras no sobrepase el tamaño del array.

aa 0
Aa 1
Aa 2
Bb 0
Bb 2
Cc 0
Cc 1
Cc 2

- a) Zero.
- b) Once.
- c) Twice
- d) Thrice

///

7. What code should be inserted?

```
4. public class Bark {  
5.     // Insert code here - Line 5  
6.     public abstract void bark();  
7. }  
8.  
9. // Insert code here - Line 9  
10.    public void bark() {  
11.        System.out.println("woof");  
12.    }  
13. }  
14. }
```

- a) 5. class Dog { 9. public class Poodle extends Dog {
- b) 5. abstract Dog { 9. public class Poodle extends Dog {
- c) 5. abstract class Dog { 9. public class Poodle extends Dog {
- d) 5. abstract Dog { 9. public class Poodle implements Dog { e)
- 5. abstract Dog { 9. public class Poodle implements Dog {
- f) 5. abstract class Dog { 9. public class Poodle implements Dog {

La clase Bark es abstracta ya que solo las clases abstractas puedes contener métodos abstractos, implicitamente sería

```
5. abstract class Dog {  
9. Public class Poodle extends Dog
```

No puede ser "implement" porque se usa para interfaces.

///

8. Which statement initializes a stringBuilder to a capacity of 128?

- a) StringBuilder sb = new String("128");
- b) StringBuilder sb = StringBuilder.setCapacity(128); C.
- c) StringBuilder sb = StringBuilder.getInstance(128); D.
- d) StringBuilder sb = new StringBuilder (128);

Para inicializar un StringBuilder

```
StringBuilder sb = new StringBuilder(128);
```

///

9. What is the result?

```
public class Calculator {  
    int num = 100;  
    public void calc(int num) {  
        this.num = num * 10;  
    }  
    public void printNum(){  
        System.out.println(num);  
    }  
    public static void main(String[] args) {  
        Calculator obj = new Calculator ();  
        obj.calc(2);  
        obj.printNum();  
    }  
}
```

El método recibe un 2 como argumento
Se multiplica $2 * 10 = 20$
Cuando llamas a printNum(), imprime el valor actual

///

10. What three modifications, made independently, made to class Greet, enable the code to compile and run?

```
package handy.dandy;
public class KeyStroke {
    public void typeExclamation() {
        System.out.println("!");
    }
}
```

And:

```
01. package handy;
02.
03.
04. public class Greet {
05.     public static void main(String[] args) {
06.         String greeting = "Hello";
07.         System.out.print(greeting);
08.         KeyStroke stroke = new KeyStroke();
09.         stroke.typeExclamation();
10.    }
11. }
```

- a) Line 8 replaced with handy.dandy.KeyStroke stroke = new KeyStroke();
- b) Line 8 replaced with handy.*.KeyStroke stroke = new KeyStroke();
- c) Line 8 replaced with handy.dandy.KeyStroke stroke = new handy.dandy.KeyStroke();
- d) import handy.*; added before line 1.
- e) import handy.dandy.*; added after line 1.
- f) import handy.dandy.KeyStroke; added after line 1.
- g) import handy.dandy.KeyStroke.typeExclamation(); added after line 1.

Las importaciones (?)

///

1. Consider the following Java code snippet:

```
public int divide (int a, int b){
    int c= -1;

    try{
        c = a/b;
    }
    catch(Exception e){
        System.err.print("Exception ");
    }
    finally{
        System.err.println("Finally ");
    }
    return c;
}
```

What will our code print when we call divide (4,0)?

- a) Exception Finally
- b) Finally Exception
- c) Exception

Al hacer una division entre 0, lanza una excepcion ArithmeticException;

Esta excepcion es capturada por catch, por lo que imprime "Exception "

Siempre se ejecuta finally por lo que el resultado seria;

Exception Finally

///

Pendiente;

3. What does the following for loop output?

```
for (int i=10, j=1; i>j; --i, ++j)
    System.out.print(j % i);
```

- a) 12321
- b) 12345
- c) 11111
- d) 00000

////

2. The feature which allows different methods to have the same name and arguments type, but the different implementation is called?

- a) Overloading(SobreCarga)
- b) Overriding (SobreEscritura @Override)
- c) Java does not permit methods with same and type signature
- d) None of the above

Permite diferentes metodos tener el mismo nombre y argumento pero con diferente implementacion; Overriding (SobreEscritura);

////

4. We perform the following sequence of actions:

1. Insert the following elements into a set: 1,2,9,1,2,3,1,4,1,5,7.
2. Convert the set into a list and sort it in ascending order.

Which option denotes the sorted list?

- a) {1, 2, 3, 4, 5, 7, 9}
- b) {9, 7, 5, 4, 3, 2, 1}
- c) {1, 1, 1, 1, 2, 2, 3, 4, 5, 7, 9}
- d) None of the above

En los Sets no puede haber duplicados, así que los elimina y el sort los ordena.

////

5. What is the output for the below Java code?

```
public class Test{  
    public static void main (String[] args)  
    {  
        int i = 010;  
        int j = 07;  
        System.out.println(i);  
        System.out.println(j);  
    }  
}
```

- a) 8 7
- b) 10 7
- c) Compilation fails with an error at line 3
- d) Compilation fails with an error at line 5

Sistema octal; llega hasta el 7.

```
1 package com.nov27.v0;  
2  
3 public class Principal {  
4  
5     public static void main(String[] args) {  
6  
7         int iDec = 11;  
8         System.out.println("Decimal: "+iDec);  
9  
10        int iBin = 0b11; //0B  
11        System.out.println("Binario: "+iBin);  
12  
13        int iOct = 011;  
14        System.out.println("Octal: "+iOct);  
15  
16        int iHex = 0x11; //0X  
17        System.out.println("Hexadecimal: "+iHex);  
18    }  
}
```

///

6. A public data member with the same name is provided in both base as well as derived classes. Which of the following is true?
- a) It is a compiler error to provide a field with the same name in both base and derived class
 - b) The program will compile and this feature is called overloading
 - c) The program will compile and this feature is called overriding
 - d) The program will compile and this feature is called as hiding or shadowing

Datos publicos con el mismo nombre se proporciona tanto en la clase base como en la clase derivada ->

El programa compilara y su caracteristica sera llamada como hiding o shadowing

///

7. Which statement is true?
- a) Non-static member classes must have either default or public accessibility
 - b) All nested classes can declare static member classes
 - c) Methods in all nested classes can be declared static
 - d) Static member classes can contain non-static methods

Las clases de miembros estáticos pueden contener métodos no estáticos

///

8. A constructor is called whenever
- a) An object is declared
 - b) An object is used
 - c) A class is declared
 - d) A class is used

Un constructor es llamada cuando – Un objeto es usado

///

9. Which of the following data types in Java are primitive?
- a) String
 - b) Struct
 - c) Boolean
 - d) Char

**Los ocho tipos primitivos definidos en Java son int, byte, short, long, float, double, boolean y char.
No se consideran objetos y representan valores en bruto**

///

10. Which of the following are true for Java Classes?

- a) The Void class extends the Class class
- b) The Float class extends the Double class
- c) The System class extends the Runtime class
- d) The Integer class extends the Number class

La clase Integer hereda de clase Number;

///

11. The following code snippet is a demonstration of a particular design pattern. Which design pattern is it?

```
public class Mystery{
    private static Mystery instance = null;
    protected Mystery(){
        public static Mystery getInstance(){
            if(instance == null){
                instance = new Mystery();
            }
            return instance;
        }
    }
}
```

- a) Factory Design Pattern
- b) Strategy Pattern
- c) Singleton
- d) Facade Design Pattern

El patrón SINGLETON garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a dicha instancia.

1. Variable estática privada
2. Constructor protegido
3. Método estático para obtener la instancia

///

12. Which of the following Java declaration of the String array is correct?

- a) String temp [] = new String {"j", "a", "z"};
- b) String temp [] = {"j" "b" "c"};
- c) String temp = {"a", "b", "c"};
- d) String temp [] = { "a", "b", "c"};

Forma correcta de declaración de un String:

String temp [] = {"a", "b", "c"};

///

13. Which is true of the following program?

```
1 package exam.java;
2
3 public class TestFirstApp {
4     static void doIt(int x, int y, int m) {
5         if(x==5) m=y;
6         else m=x;
7     }
8
9     public static void main(String[] args) {
10        int i=6, j=4, k=9;
11        TestFirstApp.doIt(i, j, k);
12        System.out.println(k);
13    }
14 }
```

- a) Doesn't matter what the values of *i* and *j* are, the output will always be 5.
- b) Doesn't matter what the values of *k* and *j* are, the output will always be 5.
- c) Doesn't matter what the values of *i* and *j* are, the output will always be 9.
- d) Doesn't matter what the values of *k* and *j* are, the output will always be 9.

J no se ve afectado. Al igual que con *k*, cualquier cambio dentro del método *doIt* afecta únicamente a las copias locales de las variables.

La lógica depende del valor de *x*, pero como *m* no afecta la variable *k*, este bloque no altera el resultado final del programa.

///

14. Which of the following statements are correct. Select the correct answer.

- a) Each Java file must have exactly one package statement to specify where the class is stored.
- b) If a java file has both import and package statement, the import statement must come before package statement.
- c) A java file has at least one class defined
- d) If a java file has a package statement, it must be the first statement (except comments).

1. Package
2. Imports
3. Classes

///

15. Given the following code, what is the most likely result.

```
import java.util.*;

public class Compares {

    public static void main(String[] args) {

        String [] cities= {"Bangalore","Pune","San Francisco","New York City"};
        MySort ms= new MySort();
        Arrays.sort(cities,ms);
        System.out.println(Arrays.binarySearch(cities,"New York City" ));
    }

    static class MySort implements Comparator{
        public int compare(String a, String b){

            return b.compareTo(a);
        }
    }
}

a) -
1 b)
1 c)
2
d) Compilation fails
```

La interfaz Comparator (java.util) en Java es genérica, es decir, debe definirse con el tipo de objetos que va a comparar. Debería ser:

Static classMySort implements Comparator <String> {

Como no se especifica el tipo genérico, el compilador arrojará un error en tiempo de compilación

El error del array es por la misma razón, intenta ordenar el arreglo cities utilizando el comparador MySort, sin embargo no compila.

///

16. To delete all pairs of keys and values in a given HashMap, which of the following methods should be used?

- a) clearAll()
- b) empty()
- c) remove()
- d) clear()

HashMap; Para eliminar todos los pares de llaves y valores debemos usar

Clear();

///

17. Which pattern do you see in the code below:

```
java.util.Calendar.getInstance();
```

- a) Singleton Pattern
- b) Factory Pattern
- c) Facade Pattern
- d) Adaptor Pattern

El método getInstance() de java.util.Calendar es un ejemplo clásico del **Factory Pattern** (Patrón de Fábrica).

- Este método no devuelve una instancia específica de una subclase de Calendar, sino que selecciona y devuelve una implementación apropiada en

función de la configuración regional y otros factores del sistema.

Java.util.Calendar.getInstance();

///

18. What is the output of the following program:

```
interface Basel { void method (); }
class BaseC
{
    public void method()
    {
        System.out.println("Inside BaseC::method");
    }
}
class ImplC extends BaseC implements Basel
{
    public static void main (String []s)
    {
        (new ImplC()).method();
    }
}
```

a) Null
b) Complicatio
fails
c) Inside
BaseC::meth
od
d) None of the
above

- Se crea un objeto de la clase **ImplC**.
- Al llamar a `(new ImplC()).method()`, se utiliza el método **method** de la clase **BaseC**, ya que no hay una sobreescritura en **ImplC**.
- Como resultado, se ejecuta el método heredado de **BaseC**, que imprime:
Inside BaseC::method

///

19. Consider the following three classes:

```
class A {}
class B extends A {}
class C extends B {}
```

Consider n object of class B is instantiated, i.e.,

`B b = new B();`

Which of the following Boolean expressions evaluates to true:

- a) `(b instanceof B)`
b) `(b instanceof B) && (!(b instanceof A))`
c) `(b instanceof B) && (!(b instanceof C))`
d) None of the above

A) Evalua si b es una instancia de la clase B

C) Evalua dos condiciones, si b es una instancia de B (true), si b NO es una instancia de C.
B no es un objeto de la clase C.

Un objeto no pertenece automáticamente a su subclase.

///

20. What is the output of the following program:

```
class Constructor
{
    static String str;
    public void Constructor(){
        System.out.println("In constructor");
        str = "Hello World";
    }
    public static void main (String [] args){
        Constructor C = new Constructor ();
        System.out.println(str);
    }
}
```

- a) In Constructor
- b) Null
- c) Compilation fails
- d) None of the above

str es una variable estática, lo que significa que pertenece a la clase y no a las instancias. Su valor inicial por defecto es null (porque es un objeto de tipo String).

No se imprime nada desde el método Constructor() porque este nunca es invocado.