Guía Java III

1.- Which declaration initializes a boolean variable?

- a) `boolean m = null`
- b) `Boolean j = (1<5)`
- c) `boolean k = 0`
- d) `boolean h = 1`

Respuesta: b) → los únicos `literal values` esperados para un boolean son `true` ó `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

Respuesta: a) → The Data Transfer Object (DTO) pattern is a software design pattern that helps to organize and manage the data exchanged between different layers of an application, making it more maintainable and efficient.

3.- What is the result?

```
int a = 10; int b = 37; int z = 0; int w = 0;
if (a==b) {z=3; } else if (a>b) {z=6;}
w = 10 * z;
```

- a) 30
- b) 0
- c) 60

Respuesta: b) → `a` es diferente de `b` por lo que no se realiza la asignación `z=3` , `a` no es mayor que `b` por lo que no se realiza la asignación `z=6` , entonces `z=0` y `w=0`

4.- Which three options correctly describe the relationship between the classes?

```
class Class1{String v1;}
class Class2{
    Class1 c1;
    String v2;
}
class Class3 {Class2 c1; String v3;}
```

- A. Class2 has-a v3.
- B. Class1 has-a v2.
- C. Class2 has-a v2.
- D. Class3 has a v1.
- E. Class2 has-a Class3.
- F. Class2 has-a Class1.

Análisis: A → La `Class2` no se encuentra relacionada con `v3` | B → La `Class1` no se encuentra relacionada con `v2` | C → `Class2` sí tiene un atributo `v2` | D → `Class3` tiene un atributo de tipo `Class2` , y `Class2` tiene un atributo de tipo `Class1`, que contiene un atributo `v1` | E → `Class2` no tiene un `Class3` | F → `Class2` sí tiene un atributo de tipo `Class1` .

Respuesta: C, D, F

5.- What is the result?

```
try {
// assume "conn" is a valid Connection object
// assume a valid Statement object is created
// assume rollback invocations will be valid
// use SQL to add 10 to a checking account
Savepoint s1 = conn.setSavePoint();
// use SQL to add 100 to the same checking account
Savepoint s2 = conn.setSavePoint();
// use SQL to add 1000 to the same checking account
// insert valid rollback method invocation here
} catch (Exception e) {}
```

1. If conn.rollback(s1) is inserted, account will be incremented by 10.
2. If conn.rollback(s1) is inserted, account will be incremented by 1010.
3. If conn.rollback(s2) is inserted, account will be incremented by 100.
4. If conn.rollback(s2) is inserted, account will be incremented by 110.
5. If conn.rollback(s2) is inserted, account will be incremented by 1110.

Explicación: `Savepoint` y `rollback` se realiza para manejar transacciones en una base de datos de manera más granular.

Si se hace rollback al primer `Savepoint` (`s1`), todas las modificaciones realizadas después de `s1` (como agregar 100 y 1000) se desharán, pero las modificaciones realizadas antes de `s1` (como agregar 10) se mantendrán.
Si se hace rollback al segundo `Savepoint` (`s2`), solo las modificaciones realizadas después de `s2` (como agregar 1000) se desharán. Las modificaciones antes de `s2` (agregar 10 y 100) se mantendrán.

6.- Which two statements are true?

A) An abstract class can implement an interface.

B) An abstract class can be extended by an interface.

C) An interface CANNOT be extended by another interface.

D) An interface can be extended by an abstract class.

E) An abstract class can be extended by a concrete class.

F) An abstract class CANNOT be extended by an abstract class.

Análisis: A → Una clase abstracta sí puede implementar una interfaz | B → Una interfaz no puede heredar de una interfaz | C → Las interfaces sólo puede heredar de interfaces | D → Las interfaces solo pueden ser extendidas por otras interfaces | E → Una clase abstracta si puede ser extendida por una clase concreta. | F → Una clase abstracta puede ser extendida por otra clase abstracta.

Respuesta: A, E

7.- Which two are possible outputs?

```
public class Main {
    public static void main(String[] args) throws Exception {
        doSomething();
    }
    private static void doSomething() throws Exception {
        System.out.println("Before if clause");
        if (Math.random() > 0.5) {
            throw new Exception();
        }
        System.out.println("After if clause");
    }
}
```

1. Before if clause Exception in thread “main” java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
2. Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15) After if clause
3. Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15)
4. Before if clause After if clause

Explicación: La opción **1** se produce cuando se evalúa en el if si el valor generado por `Math.random()` es mayor que 0.5. En ese caso, se lanza una excepción y no se ejecuta la línea `System.out.println("After if clause");`. Por lo tanto, solo se imprime `"Before if clause"` seguido del stack trace de la excepción.

La opción 4 se produce cuando el valor generado por `Math.random()` es menor o igual a 0.5. Debido a que no entra en el if, no se lanza una excepción y ambas líneas `System.out.println("Before if clause");` y `System.out.println("After if clause");` se ejecutan.

Respuesta: 1,4

8.- 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-=1) {
            jj++;
        }
        System.out.println("jj = " + jj);
    }
}
```

1. -1
2. 1
3. 5
4. 8
5. 11

Análisis: `kk` determina el inicio del ciclo `for` , el ciclo va en decremento y dentro se incrementa `jj` . Para 11 el número de iteraciones que se realizan son las correspondientes a 11,10,9,8 y 7, imprimiendo `jj=5` .

Respuesta: 5

9.- What will make this code compile and run?

```
public class Simple {

    public float price;
    public static void main(String[] args) {

        Simple price = new Simple();
        price = 4;
    }
}
```

1. Change line 3 to the following: public int price;
2. Change line 7 to the following: int price = new Simple();
3. Change line 7 to the following: float price = new Simple();
4. Change line 7 to the following: price = 4f;
5. Change line 7 to the following: price.price = 4;

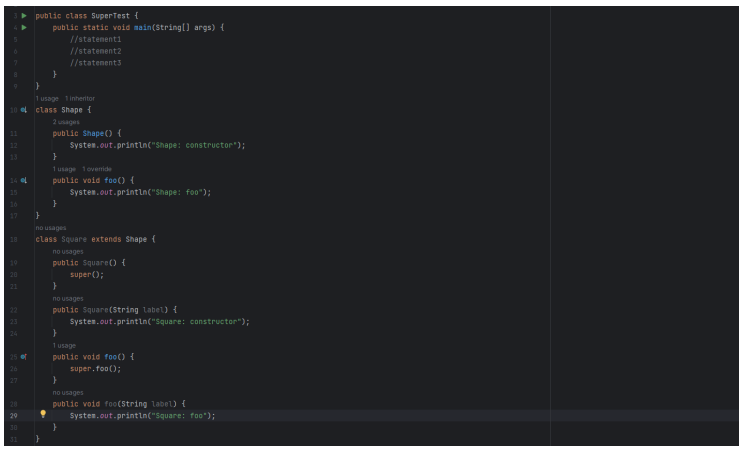
Respuesta: 5) → Si quisiéramos cambiar la referencia de el price de la clase, tendríamos que ingresar al objeto para hacerlo, de otra forma estamos intentando cambiar la referencia de Simple price, y 4 (int) no es de la clase Simple.

10.- In the Java collections framework a Set Is:

1. A collection that cannot contain duplicate elements.
2. An ordered collection that can contain duplicate elements.
3. An object that maps value key sets and cannot contain values Duplicates.

Respuesta → 1) Un Set pertenece a collection y no permite el tener elementos duplicados en su interior. Los otros dos puntos hacen referencia a List y Map, respectivamente.

11.- What should statement1, statement2, and statement3, be respectively, in order to produce the result?



Shape: constructor

Shape: foo

Square: foo

1. Square square = new Square("bar"); square.foo("bar"); square.foo();
2. Square square = new Square("bar"); square.foo("bar"); square.foo("bar");
3. Square square = new Square(); square.foo(); square.foo(bar);
4. Square square = new Square(); square.foo(); square.foo("bar");
5. Square square = new Square(); square.foo(); square.foo();

Respuesta: 4) Solo en la opción 4 obtenemos el resultado que buscamos, ya que:

Square square = new Square();: Imprime "Shape: constructor".

square.foo();: Imprime "Shape: foo".

square.foo("bar");: Imprime "Square: foo".

12.- What is the result?

```
public class SampleClass {
    public static void main(String[] args) {
        AnotherSampleClass asc = new AnotherSampleClass();
        SampleClass sc = new SampleClass();
        sc = asc;
        System.out.println("sc: " + sc.getClass());
        System.out.println("asc: " + asc.getClass());
    }
}

class AnotherSampleClass extends SampleClass { }
```

A. sc: class.Object asc: class.AnotherSampleClass

B. sc: class.SampleClass asc: class.AnotherSampleClass

C. sc: class.AnotherSampleClass asc: class.SampleClass

D. sc: class.AnotherSampleClass asc: class.AnotherSampleClass

Respuesta: D) → `getClass()` evalúa la clase de el objeto, y no de la variable de referencia, por lo que al pasarle a sc el objeto `AnotherSampleClass()` , nos da este resultado.

---

13. What is true about the class Wow?

```
public abstract class Wow {
    private int wow;
    public Wow(int wow) { this.wow = wow; }
    public void wow() { }
    private void wowza() { }
}
```

A. It compiles without error.

B. It does not compile because an abstract class cannot have private methods.

C. It does not compile because an abstract class cannot have instance variables.

D. It does not compile because an abstract class must have at least one abstract method.

E. It does not compile because an abstract class must have a constructor with no arguments.

Respuesta: *A)* → Una clase abstracta puede tener constructores, variables de instancia, métodos privados y no abstractos.

---

14. The SINGLETON pattern allows:

A. Have a single instance of a class and this instance cannot be used by other classes.

B. Having a single instance of a class, while allowing all classes have access to that instance.

C. Having a single instance of a class that can only be accessed by the first methods that calls it.

Respuesta: *B) → Having a single instance of a class, while allowing all classes have access to that instance.*

---

15. How many times is 2 printed?

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

A. Zero.

B. Once.

C. Twice.

D. Thrice.

E. It is not printed because compilation fails.

Respuesta: *B)* → Dentro del `foreach` tenemos un ciclo que implementa un `break` sin condición por lo que el ciclo `while` no continuará después de imprimir e incrementar el valor de `ii`, el valor de `ii` se incrementará hasta 2, debido a la expresión booleana dada en el ciclo. Así que sólo una vez tendrá el valor de 2 y solo se podrá imprimir una vez.

16.What is the result?

```
public static void main(String[] args) {
    int [][] array2D = { {0, 1, 2}, {3, 4, 5, 6} };
    System.out.print(array2D[0].length + "");
    System.out.print(array2D[1].getClass().isArray() + "" );
    System.out.println(array2D[0][1]);
}
```

A. 3false1

B. 2true3

C. 2false3

D. 3true1

E. 3false3

F. 2true1

G. 2false1

Respuesta: *D)* → En la primera impresión se obtiene el tamaño del primer arreglo en `array2D` que es de 3, la segunda impresión obtiene la clase a la que pertenece el segundo arreglo almacenado en `array2D`, y verifica si es de tipo `Array`, por lo que se obtiene `true`, y la última impresión imprime el entero almacenado en el primer arreglo que se encuentra en el índice 1, y corresponde a 1.

17.- In Java the difference between throws and throw is:

1. Throws throws an exception and throw indicates the type of exception that the method.
2. Throws is used in methods and throw in constructors.
3. Throws indicates the type of exception that the method does not handle and throw an exception.

Respuesta: 3) → `throw` se utiliza para lanzar una excepción en el cuerpo de un método o constructor, `throws` se utiliza en la declaración de un método para indicar que el método puede lanzar una excepción y que la excepción debe ser manejada por el código que llama al método.

18.- What is the result?

```
class Person {
String name = "No name";
public Person (String nm) {name=nm}
}

class Employee extends Person{
    String empID = "0000";
    public Employee(String id) { empID " //18
}
}

public class EmployeeTest {
    public static void main(String[] args) {
        Employee e = new Employee("4321");
        System.out.printiln(e.empID);
    }
}
```

1. 4321.
2. 0000.
3. An exception is thrown at runtime.
4. Compilation fails because of an error in line 18.

Respuesta: 4) → El código no compila debido a errores en la escritura, desde que falta un `;` en el constructor de `Person`, hasta caracteres sin sentido en el constructor del `Employee`

19.- Which is true?

```
class Building {}
    public class Barn extends Building{
        public static void main(String[] args){
            Building build1 = new Building();
            Barn barn1 = new Barn();
            Barn barn2 = (Barn) build1; //10
            Object obj1 = (Object) build1; //11
```

```
            String str1 = (String) build1; //12
            Building build2 = (Building) barn1; //13
        }
    }
```

1. If line 10 is removed, the compilation succeeds.
2. If line 11 is removed, the compilation succeeds.
3. If line 12 is removed, the compilation succeeds.
4. If line 13 is removed, the compilation succeeds.
5. More than one line must be removed for compilation to succeed.

Respuesta: 3) → La clase `String` no tiene relación alguna con `Building` , por lo que el cast no se realiza y ni compila el código.

20.- What is the result?

```
class Atom {
    Atom() {System.out.print("atom ");}
}

class Rock extends Atom {
    Rock(String type) {System.out.print(type);}
}

public class Mountain extends Rock {
    Mountain(){
        super("granite ");
        new Rock("granite ");
    }
    public static void main(String[] a) {new Mountain();}
}
```

1. Compilation fails.
2. Atom granite.
3. Granite granite.
4. Atom granite granite.
5. An exception is thrown at runtime.
6. Atom granite atom granite.

Respuesta: 6) → Se inicia creando una instancia de `Mountain` , por lo que se llama a su constructor, que a su vez llama al constructor de `Rock` con el valor `"granite "` . El constructor de `Rock` invoca al constructor de `Atom` , por lo que la primera impresión es `"atom"` , posteriormente se ejecuta la línea que imprime en el constructor de `Rock` y se imprime `"granite "` . Luego se crea un `Rock` y se pasa como argumento `"granite "` , por lo que el ciclo se repite y finalmente se obtiene `"atom granite atom granite"`

21.- Which statement is true?

```
class ClassA {
    public int numberOfInstances;
```

```
    protected ClassA(int numberOfInstances) {
        this.numberOfInstances = numberOfInstances;
    }
}

public class ExtendedA extends ClassA {
        private ExtendedA(int numberOfInstances) {
            super(numberOfInstances);
        }

        public static void main(String[] args) {
            ExtendedA ext = new ExtendedA(420);
            System.out.print(ext.numberOfInstances);
        }
}
```

A. 420 is the output.

B. An exception is thrown at runtime.

C. All constructors must be declared public.

D. Constructors CANNOT use the private modifier.

E. Constructors CANNOT use the protected modifier.

Respuesta: A) → El programa se ejecuta adecuadamente, no hay una excepción, los constructores pueden utilizar todos los modificadores de acceso.

22.- What is the result?

```
public class Test {
    public static void main(String[] args) {
        int[][] array = { {0}, {0,1}, {0,2,4}, {0,3,6,9}, {0,4,8,12,16} };
        System.out.println(array[4][1]);
        System.out.println(array[1][4]);
    }
}
```

A. 4 Null.

B. Null 4.

C. An IllegalArgumentException is thrown at run time.

D. 4 An ArrayIndexOutOfBoundsException is thrown at run time.

Respuesta: D) → Se imprime el 4 y posteriormente se intenta acceder a una posición que excede el tamaño del arreglo.

23.- What is the result?

```
public class X {
    public static void main(String[] args) {
        String theString = "Hello World";
        System.out.println(theString.charAt(11));
```

```
    }
}
```

A. There is no output.

B. d is output.

C. A StringIndexOutOfBoundsException is thrown at runtime.

D. An ArrayIndexOutOfBoundsException is thrown at runtime.

E. A NullPointException is thrown at runtime.

F. A StringArrayIndexOutOfBoundsException is thrown at runtime.

Respuesta: → C) Es la excepción que se lanza cuando intentamos acceder a un caractér que supera el tamaño de la cadena.

24.- What is the result?

```
public class Bees {
    public static void main(String[] args) {
        try {
            new Bees().go();
        }catch (Exception e) {
            System.out.println("thrown to main");
        }
    }

    synchronized void go() throws InterruptedException {
        Thread t1 = new Thread();
        t1.start();
        System.out.print("1 ");
        t1.wait(5000);
        System.out.print("2 ");
    }
}
```

A. The program prints 1 then 2 after 5 seconds.

B. The program prints: 1 thrown to main.

C. The program prints: 1 2 thrown to main.

D. The program prints: 1 then t1 waits for its notification.

Puntos claves:

- InterruptedException pertenece a java.lang y java.lang se importa automáticamente.
- El método .wait() es quien lanza la checked exception InterruptedException.
- El orden de ejecución de los hilos es decisión de la JVM y se puede ejecutar en cualquier orden.
- En un catch es posible cachar al padre de la excepción
- synchronized asegura que únicamente un hilo pueda ejecutar un método a la vez en una instancia.
- Instancia de un hilo (Thread t1 = new Thread();), un hilo es una tarea separada que puede ejecutarse concurrentemente con otras tareas.
- El método .start() indica que un hilo comienza a ejecutarse.

- .wait() hace que el hilo actual espere y libere el monitor permitiendo que otros hilos lo adquieran..
- En programación concurrente un monitor es el mecanismo que solo un hilo a la vez pueda ejecutar un bloque de código. El monitor es como una llave que asegura que solo un hilo puede entrar a un bloque de código a la vez.
- En Java, los monitores están asociados con cada objeto y se utilizan para gestionar la sincronización.
- Cuando un hilo quiere ejecutar un bloque synchronized necesita adquirir el monitor del objeto. Si otro hilo ya tiene al monitor, el hilo actual debe esperar hasta que el monitor esté disponible.
- Cuando el hilo termina su ejecución libera el monitor, permitiendo que otros hilos puedan adquirirlo.
-

Análisis de código

- En el método main se crea una instancia de Bees y se llama al método go().
- El método go() es synchronized, lo que significa que cualquier hilo que lo llame debe adquirir el monitor del objeto Bees antes de ejecutarlo.
- Se crea un nuevo hilo y se inicia aunque no se le da ninguna tarea a ejecutar.
- Imprime 1.
- Intenta que el hilo t1 espere 5 segundos. Esto falla porque el hilo principal posee el monitor del objeto Bees, no de t1, esto causa un IllegalMonitorStateException.
- El programa ejecuta el catch, imprime y termina.

Respuesta correcta:

The program prints: 1 thrown to main.

Notas

A pesar que en el método go() se indica el lanzamiento de InterruptedException, la excepción que en realidad de lanza es IllegalMonitorStateException. El manejo de la excepción se logra debido a que en el catch se maneja Exception.

25.- ¿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;
    }
}
```

1. Compilation fails
2. An exception is thrown at runtime
3. There is no result because this is not correct way to determine the hash code
4. Hash is: 111111, 44444444, 999999999.

Respuesta: 4) → No existen errores ni excepciones, y el método no tiene relación con los hashcode, sin embargo no se tiene contexto de la funcionalidad.

26.- ¿Cuál sería el resultado?

```
public class Test {
    public static void main(String[] args) {
        int b = 4;
        b--;
        System.out.println(--b);
        System.out.println(b);
    }
}
```

1. 22
2. 12
3. 32
4. 33

Respuesta: 1) → Se decrementa a `b` , se decrementa a `b` nuevamente y luego se imprime `2` , se imprime nuevamente `b` .

27.- ¿Cuál sería el resultado?

```
import java.util.*;
public class App {
    public static void main(String[] args) {
        List p = new ArrayList();
        p.add(7);
        p.add(1);
        p.add(5);
        p.add(1);
        p.remove(1);
        System.out.println(p);
    }
}
```

1. [7, 1, 5, 1]
2. [7, 5, 1]
3. [7, 5]
4. [7, 1]}

Respuesta: 2) → El método `remove()` indica que se remueva el elemento de índice 1 de la lista.

28.- ¿Cuál sería el resultado?

```
public classDoCompare4 {
    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);
    }
}
```

1. 0
2. 0 1 2
3. 0 1 2 0 1 2 0 1 2
4. Compilation fails

Respuesta: 2) → El tamaño de `table` es 3, la primera impresión es 0 y se incrementa a `ii` , la segunda impresión es 1 y se incrementa a `ii` , la tercera impresión es 2 y se incrementa a `ii` , el ciclo `while` termina y se evalúa el `do-while` , donde `ii=3` por lo que no se ejecuta de nuevo.

29.- ¿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;9
            while (ii < table.length) {
                System.out.println(ss + ", " + ii);
                ii++;
            }
        }
    }
}
```

1. Zero.
2. Once.
3. Twince
4. Thrice

5. Compilation fails

Respuesta: 4) → El resultado final de las impresiones será:aa, 2 -bb, 2 - cc, 2 = 3 veces sale el 2.

---

30. What is the result?

```
public class Boxer1 {
    Integer i;
    int x;

    public Boxer1(int y) {
        x = i + y;
        System.out.println(x);
    }

    public static void main(String[] args) {
        new Boxer1(new Integer(4));
    }
}
```

A. The value “4” is printed at the command line.

B. Compilation fails because of an error in line 5.

C. Compilation fails because of an error in line 9.

D. A NullPointerException occurs at runtime.

E. A NumberFormatException occurs at runtime.

F. An IllegalStateException occurs at runtime.

Respuesta: D) → `i` almacena `null` , pues es un `Integer` .

---

31. What is the result?

A. Class Base1 { abstract class Abs1 { } }

B. Abstract class Abs2 { void doit() { } }

C. class Base2 { abstract class Abs3 extends Base2 { } }

D. class Base3 { abstract int var1 = 89; }

- Esto es ilegal.

Respuesta D) → Es posible tener una clase abstracta dentro de otra clase, una clase abstracta puede tener métodos concretos, es posible tener una clase abstracta que extiende otra clase dentro de una clase, las variables no pueden ser abstractas. La palabra clave abstract solo se aplica a métodos y clases.

32.- ¿Cuál sería el resultado?

```
public class ScopeTest {
    int z;
    public static void main(String[] args) {
        ScopeTest myScope = new ScopeTest();
        int z = 6;
        System.out.print(z);
        myScope.doStuff();
        System.out.print(z);
```

```
        System.out.print(myScope.z);
    }

    void doStuff() {
        int z = 5;
        doStuff2();
        System.out.print(z);
    }

    void doStuff2() {
        z = 4;
    }
}
```

1. **6564**
2. 6554
3. 6566
4. 6565

Para confirmar nuestra comprensión:

1. **System.out.print(z); en main imprime 6.**
2. **myScope.doStuff(); llama a doStuff:**
   - **doStuff declara int z = 5;.**
   - **doStuff2 establece myScope.z = 4;.**
   - **System.out.print(z); en doStuff imprime 5.**
3. **System.out.print(z); en main después de doStuff imprime 6.**
4. **System.out.print(myScope.z); imprime 4.**

Entonces, el orden de las impresiones y sus valores será 6564.

Respuesta: 1)

49-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();
```

```
    }
}
```

a-20

b-100

c-1000

d-2

Respuesta: a) → 20 por el flujo del código

50.- ¿Qué tres modificaciones, hechas de manera independiente, permitirán que la clase Greet compile y se ejecute?

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

package handy;
public class Greet {
    public static void main(String[] args) {
        String greeting = "Hello";
        System.out.print(greeting);
        KeyStroke stroke = new KeyStroke();
        stroke.typeExclamation();
    }
}
```

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.

52-What is the result?

```
class Feline {
    public String type = "f ";
    public Feline() {
        System.out.print("feline ");
    }
}

public class Cougar extends Feline {
    public Cougar() {
```

```
        System.out.print("cougar ");
    }
    void go() {
        type = "c ";
        System.out.print(this.type + super.type);
    }
    public static void main(String[] args) {
        new Cougar().go();
    }
}
```

A. Cougar c f.

B. Feline cougar c c.

C. Feline cougar c f.

D. Compilation fails.

53-What is the result?

```
interface Rideable {
    String getGait();
}

public class Camel implements Rideable {
    int weight = 2;

    String getGait() {
        return mph + ", lope"; // Error de compilación: mph no está definido
    }

    void go(int speed) {
        ++speed;
        weight++;
        int walkrate = speed * weight;
        System.out.print(walkrate + getGait());
    }
    public static void main(String[] args) {
        new Camel().go(8);
    }
}
```

A. 16 mph, lope.

B. 24 mph, lope.

C. 27 mph, lope.

D. Compilation fails.

54.- ¿Cuáles de las siguientes opciones son instanciaciones e inicializaciones válidas de un arreglo multidimensional?

A.- int[][] array2D = {{0, 1, 2, 4}{5, 6}};

int[][] array2D = new int[2][2];

B.- array2D[0][0] = 1;

array2D[0][1] = 2;

array2D[1][0] = 3;

array2D[1][1] = 4;

int[] array3D = new int[2][2][2];

C.- int[][] array3D = {{{0, 1}, {2, 3}, {4, 5}}};

int[] array = {0, 1};

D.-array3D[0][0] = array;

array3D[0][1] = array;

array3D[1][0] = array;

array3D[1][1] = array;

Extraído de capture4-preguntas