

Guía JAVA

1. Excepciones de Java permitida*

- `org.springframework.web.client.RestClientException` para manejar errores en solicitudes HTTP a través del cliente `RestTemplate`.
- `No conozco la respuesta`
- `java.lang.NumberFormatException` cuando se intenta convertir una cadena en un número no es válido
- `com.bbva.apx.exception.db.NoResultException` Una excepción personalizada BBVA
- `com.bbva.elara.utility.interbackend.cics.exceptions.BusinessException` Excepción personalizada para lógica de negocio BBVA

2. Which three are bad practices?

- Checking for `ArrayIndexOutOfBoundsException` and ensuring that the program can recover if one occurs.
- Checking for `FileNotFoundException` to inform a user that a filename entered is not valid.
- Checking for Error and, if necessary, restarting the program to ensure that users are unaware problems.
- Checking for `ArrayIndexOutOfBoundsException` when iterating through an array to determine when all elements have been visited.

3. Indica a JUnit que la propiedad que usa esta anotación es una simulación y, por lo tanto, se inicializa como tal y es susceptible de ser inyectada por `@InjectMocks`.

- Mockito
- **Mock** @Mock: Es una anotación proporcionada por Mockito para simular un objeto. Permite crear dependencias simuladas que pueden ser controladas en pruebas unitarias.
- Inject
- InjectMock

4.

Given

```
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.print(array2D[0][1]);  
}
```

What is the result?

- 3false3
- 3false1
- 2false1
- **3true1**
- 2true3

5. Which two statements are true?

- An interface CANNOT be extended by another interface.
- **An abstract class can be extended by a concrete class.**
- An abstract class CANNOT be extended by an abstract class. An interface can be extended by an abstract class.
- **An abstract class can implement an interface**
- An abstract class can be extended by an interface.

6. Which five methods, inserted independently at line 5, will compile? (Choose five)

```
1 public class Blip{
2     protected int blipvert(int x){ return 0
3 }
4 class Vert extends Blip{
5     //insert code here
6 }
```

- Private int blipvert(long x) { return 0; }
- Protected int blipvert(long x) { return 0; }
- Protected long blipvert(int x, int y) { return 0; }
- Public int blipvert(int x) { return 0; }
- Private int blipvert(int x) { return 0; }
- Protected long blipvert(int x) { return 0; }
- Protected long blipvert(long x) { return 0; }

7.

Given:

```
1. class Super{
2.     private int a;
3.     protected Super(int a){ this.a = a; }
4. }
...
11. class Sub extends Super{
12.     public Sub(int a){ super(a);}
13.     public Sub(){ this.a = 5;}
14. }
```

Which two independently, will allow Sub to compile? (Choose two)

- Change line 2 to: public int a;
- Change line 13 to: public Sub(){ super(5);}
- Change line 2 to: protected int a;
- Change line 13 to: public Sub(){ this(5);}
- Change line 13 to: public Sub(){ super(a)}

8. Whats 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() { }  
}
```

- It compiles without error.
- It does not compile because an abstract class cannot have private methods
- It does not compile because an abstract class cannot have instance variables.
- It does not compile because an abstract class must have at least one abstract method.
- It does not compile because an abstract class must have a constructor with no arguments.

Clases abstractas:

- Una clase abstracta es aquella que no puede ser instanciada directamente.
- Puede contener métodos concretos (con cuerpo), métodos abstractos (sin cuerpo) o ambos.

Modificadores en clases abstractas:

- Las clases abstractas pueden tener métodos y variables con cualquier nivel de acceso (public, protected, private).
- Los métodos privados solo son accesibles dentro de la misma clase, independientemente de si la clase es abstracta o no.

Constructores en clases abstractas:

- Las clases abstractas pueden tener constructores, y estos pueden ser llamados desde las subclases usando super.

Requisitos para métodos abstractos:

- Si una clase tiene al menos un método abstracto, la clase debe ser marcada como abstracta.
- Una clase abstracta no está obligada a tener métodos abstractos.

9. 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(); }  
}
```

- Compilation fails.
- Atom granite.
- Granite granite.
- Atom granite granite.
- An exception is thrown at runtime
- Atom granite atom granite

10. What is printed out when the program is executed?

```
public class MainMethod {  
    void main() {  
        System.out.println("one");  
    }  
    static void main(String args) {  
        System.out.println("two");  
    }  
    public static final void main(String[] args) {  
        System.out.println("three");  
    }  
    void mina(Object[] args) {  
        System.out.println("four");  
    }  
}
```

-- Este es el método main principal que se ejecuta, los demás son métodos comunes que no han sido llamados para ser ejecutados

- one
- two
- **three**
- four
- There is no output

11. 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();  
    }  
}
```

Cuando se crea una instancia de Cougar con new Cougar(), primero se ejecuta el constructor de la clase base Feline antes de ejecutar el constructor de Cougar

- Cougar c f.
- Feline cougar c f.
- **Feline cougar c c.**
- Compilation fails.

12. What is the result?

```
class Alpha { String getType() { return "alpha"; } }  
class Beta extends Alpha { String getType() { return "beta"; } }  
public class Gamma extends Beta { String getType() { return "gamma"; }  
    public static void main(String[] args) {  
        Gamma g1 = new Alpha();  
        Gamma g2 = new Beta();  
        System.out.println(g1.getType() + " " + g2.getType());  
    }  
}
```

se intenta asignar una instancia de Alpha a una variable de tipo Gamma y sucede lo mismo con Beta

De lado izquierdo siempre deben ir clases padres / abuelos - lado derecho hijos/nietos

- Alpha beta
- Beta beta.
- Gamma gamma.
- **Compilation fails.**

13. What is the result?

```
import java.util.*;
public class MyScan {
    public static void main(String[] args) {
        String in = "1 a 10 . 100 1000";
        Scanner s = new Scanner(in);
        int accum = 0;
        for (int x = 0; x < 4; x++) {
            accum += s.nextInt();
        }
        System.out.println(accum);
    }
}
```

El método nextInt() lanza una excepción InputMismatchException si el siguiente token no es un número entero.

- 11
- 11
- 1111
- An exception is thrown at runtime InputMismatchException

14. 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 ");  
    }  
}
```

- The program prints 1 then 2 after 5 seconds.
- **The program prints: 1 thrown to main.**
- The program prints: 1 2 thrown to main.
- The program prints:1 then t1 waits for its notification.

15. 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);
    }
}
```

• **420 is the output.**

- An exception is thrown at runtime.
- All constructors must be declared public.
- Constructors CANNOT use the private modifier.
- Constructors CANNOT use the protected modifier

Esto es falso, los constructores pueden tener cualquier nivel de acceso (public, protected, private o package-private). En este caso, el constructor de ExtendedA es privado, lo cual es válido.

Esto es incorrecto. Los constructores privados son válidos y se usan comúnmente, por ejemplo, en patrones como el Singleton.

También es falso. Los constructores protegidos son válidos y permiten que solo las clases dentro del mismo paquete o las clases derivadas puedan acceder a ellos. En este caso, ClassA tiene un constructor protegido que es llamado desde su subclase.

16. The SINGLETON pattern allows:

- Have a single instance of a class and this instance cannot be used by other classes
- Having a single instance of a class, while allowing all classes have access to that instance.
- Having a single instance of a class that can only be accessed by the first method that calls it.

--- El patrón Singleton asegura que solo haya una única instancia de una clase durante el tiempo de vida de la aplicación.

--- Esta instancia es accesible globalmente a través de un método estático (generalmente llamado `getInstance()`), permitiendo que todas las clases puedan utilizarla.

17. What is the result?

```
import java.text.*;
public class Align {
    public static void main(String[] args) throws ParseException {
        String[] sa = {"111.234", "222.5678"};
        NumberFormat nf = NumberFormat.getInstance();
        nf.setMaximumFractionDigits(3);
        for (String s : sa) { System.out.println(nf.parse(s)); }
    }
}
```

- 111.234 222.567
- 111.234 222.568
- 111.234 222.5678
- An exception is thrown at runtime.

18. What is the result?

Given

```
public class SuperTest {  
    public static void main(String[] args) {  
        //statement1  
        //statement2  
        //statement3  
    }  
}  
class Shape {  
    public Shape() {  
        System.out.println("Shape: constructor");  
    }  
    public void foo() {  
        System.out.println("Shape: foo");  
    }  
}  
class Square extends Shape {  
    public Square() {  
        super();  
    }  
    public Square(String label) {  
        System.out.println("Square: constructor");  
    }  
    public void foo() {  
        super.foo();  
    }  
    public void foo(String label) {  
        System.out.println("Square: foo");  
    }  
}
```

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

Shape: constructor
Shape: foo
Square: foo

- Square square = new Square ("bar"); square.foo ("bar"); square.foo();
- Square square = new Square ("bar"); square.foo ("bar"); square.foo ("bar");
- Square square = new Square (); square.foo (); square.foo(bar);
- **Square square = new Square (); square.foo (); square.foo("bar");**
- Square square = new Square (); square.foo (); square.foo ();

19. Which three implementations are valid?

```
interface SampleCloseable {  
    public void close() throws java.io.IOException;  
}
```

- `class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something }}`
- `class Test implements SampleCloseable { public void close() throws Exception { // do something }}`
- `class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something }}`
- `class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something }}`
- `class Test implements SampleCloseable { public void close() { // do something }}`

`public void close() throws java.io.IOException`

La implementación respeta exactamente la excepción declarada en la interfaz. Es válido declarar la misma excepción.

`public void close() throws FileNotFoundException`

`FileNotFoundException` es una subclase de `java.io.IOException`. Esto es válido porque la regla permite declarar excepciones más específicas que la original.

`public void close()`

No lanzar ninguna excepción (o no declararla) es válido, ya que un método sobrescrito puede eliminar la declaración de excepciones.

20. What is the result?

```
class MyKeys {  
    Integer key;  
    MyKeys(Integer k) { key = k; }  
    public boolean equals(Object o) {  
        return ((MyKeys) o).key == this.key;  
    }  
}
```

And this code snippet:

```
Map m = new HashMap();  
MyKeys m1 = new MyKeys(1);  
MyKeys m2 = new MyKeys(2);  
MyKeys m3 = new MyKeys(1);  
MyKeys m4 = new MyKeys(new Integer(2));  
m.put(m1, "car");  
m.put(m2, "boat");  
m.put(m3, "plane");  
m.put(m4, "bus");  
System.out.print(m.size());
```

- 2
- 3
- 4
- Compilation fails.

21. What value of x, y, z will produce the following result?

1234,1234,1234 ----, 1234, -----

```
public static void main(String[] args) {  
    // insert code here  
    int j = 0, k = 0;  
    for (int i = 0; i < x; i++) {  
        do {  
            k = 0;  
            while (k < z) {  
                k++;  
                System.out.print(k + " ");  
            }  
            System.out.println(" ");  
            j++;  
        } while (j < y);  
        System.out.println("----");  
    }  
}
```

- int x=4, y=3, z=2;
- int x=3, y=2, z=3;
- int x=2, y=3, z=3;
- **int x=2, y=3, z=4;**
- int x=4, y=2, z=3;

22. Which three lines will compile and output "Right on!"?

```
13. public class Speak {  
14.     public static void main(String[] args) {  
15.         Speak speakIT = new Tell();  
16.         Tell tellIt = new Tell();  
17.         speakIT.tellItLikeltIs();  
18.         (Truth) speakIT.tellItLikeltIs();  
19.         ((Truth) speakIT).tellItLikeltIs();  
20.         tellIt.tellItLikeltIs();  
21.         (Truth) tellIt.tellItLikeltIs();  
22.         ((Truth) tellIt).tellItLikeltIs();  
23.     }  
24. }
```

```
class Tell extends Speak implements Truth {  
    @Override  
    public void tellItLikeltIs() {  
        System.out.println("Right on!");  
    }  
}
```

```
interface Truth {  
    public void tellItLikeltIs();  
}
```

- Line 17
- Line 18
- Line 19
- Line 20
- Line 21
- Line 22

23. ¿Cuál es 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);
    }
}
```

- [7, 5]
- [7, 1]
- [7, 5, 1]
- [7, 1, 5, 1]

24. What is the result?

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

- 22
- 12
- 32
- 33

25. In Java the difference between throws and throw is:

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

Diferencias clave entre throw y throws :		
Aspecto	throw	throws
Propósito	Lanza una excepción específica.	Declara excepciones que un método puede lanzar.
Ubicación	Dentro del cuerpo de un método o constructor.	En la firma del método.
Uso	Ejecuta la excepción en tiempo de ejecución.	Indica responsabilidad de manejo de excepciones.

26. Which statement, when inserted into line " // TODO code application logic here", is valid in compilation time change?

```
public class SampleClass {  
    public static void main(String[] args) {  
        AnotherSampleClass asc = new AnotherSampleClass();  
        SampleClass sc = new SampleClass();  
        // TODO code application logic here  
    }  
}  
class AnotherSampleClass extends SampleClass { }
```

- asc = sc;
- **sc = asc;**
- asc = (Object) sc;
- asc = sc.clone();

27. 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]);  
    }  
}
```

- 4 Null.
- Null 4.
- An IllegalArgumentException is thrown at run time.
- **4 An ArrayIndexOutOfBoundsException is thrown at run time.** ArrayIndexOutOfBoundsException

28. Which three are valid? (Choose three)

```
class ClassA {}  
class ClassB extends ClassA {}  
class ClassC extends ClassA {}
```

And:

```
ClassA p0 = new ClassA();  
ClassB p1 = new ClassB();  
ClassC p2 = new ClassC();  
ClassA p3 = new ClassB();  
ClassA p4 = new ClassC();
```

- `p0 = p1;`
- `p1 = p2;`
- `p2 = p4;`
- `p2 = (ClassC)p1;`
- `p1 = (ClassB)p3;`
- `p2 = (ClassC)p4;`

29. 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; }
```

- Class2 has-a v3.
- Class1 has-a v2.
- Class2 has-a v2.
- Class3 has-a v1.
- Class2 has-a Class3
- Class2 has-aClass1

30. What is the result?

```
class MySort implements Comparator<Integer> {  
    public int compare(Integer x, Integer y) {  
        return y.compareTo(x);  
    }  
}
```

And the code fragment:

```
Integer[] primes = {2, 7, 5, 3};  
MySort ms = new MySort();  
Arrays.sort(primes, ms);  
for (Integer p2 : primes) { System.out.print(p2 + " "); }
```

- 2 3 5 7
- 2 7 5 3
- 7 5 3 2
- Compilation fails.

31. Which two 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");  
    }  
}
```

- Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
- 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.
- Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
- Before if clause After if clause.

32. What is the result?

```
public static void main(String[] args) {  
    String color = "Red";  
    switch (color) {  
        case "Red":  
            System.out.println("Found Red");  
        case "Blue":  
            System.out.println("Found Blue");  
        case "White":  
            System.out.println("Found White");  
            break;  
        Default:  
            System.out.println("Found Default");  
    }  
}
```

- Found Red.
- Found Red Found Blue.
- **Found Red Found Blue Found White.**
- Found Red Found Blue Found White Found Default.

33. What is the result?

```
class X {  
    static void m(int i) {  
        i += 7;  
    }  
    public static void main(String[] args) {  
        int i = 12;  
        m(i);  
        System.out.println(i);  
    }  
}
```

- 7
- **12**
- 19
- Compilation fails.
- An exception is thrown at run time

34. Which is true?

```
5. class Building { }
6.     public class Barn extends Building {
7.         public static void main(String[] args) {
8.             Building build1 = new Building();
9.             Barn barn1 = new Barn();
10.            Barn barn2 = (Barn) build1;
11.            Object obj1 = (Object) build1;
12.            String str1 = (String) build1;
13.            Building build2 = (Building) barn1;
14.        }
15.    }
```

Which is true?

- If line 10 is removed, the compilation succeeds.
- If line 11 is removed, the compilation succeeds.
- **If line 12 is removed, the compilation succeeds.**
- If line 13 is removed, the compilation succeeds.
- More than one line must be removed for compilation to succeed.

35. What is the result if the integer value is 33?

```
public static void main(String[] args) {  
    if (value >= 0) {  
        if (value != 0) {  
            System.out.print("the ");  
        } else {  
            System.out.print("quick ");  
        }  
        if (value < 10) {  
            System.out.print("brown ");  
        }  
        if (value > 30) {  
            System.out.print("fox ");  
        } else if (value < 50) {  
            System.out.print("jumps ");  
        } else if (value < 10) {  
            System.out.print("over ");  
        } else {  
            System.out.print("the ");  
        }  
        if (value > 10) {  
            System.out.print("lazy ");  
        } else {  
            System.out.print("dog ");  
        }  
        System.out.print("... ");  
    }  
}
```

- The fox jump lazy...
- The fox lazy...
- Quick fox over lazy...
- Quick fox the...

36. What is the result?

```
11. class Person {
12.     String name = "No name";
13.     public Person (String nm) { name = nm}
14. }
15.
16. class Employee extends Person {
17.     String empID = "0000";
18.     public Employee (String id) { empID "
19.         id; }
20. }
21. public class EmployeeTest {
22.     public static void main(String[] args)
23.     {
24.         Employee e = new Employee("4321");
25.         System.out.println(e.empID);
26.     }
27. }
```

- 4321.
- 0000.
- An exception is thrown at runtime.
- **Compilation fails because of an error in line 18.**

37. Which code fragment is illegal?

- Class Base1 { abstract class Abs1 { }}
- Abstract class Abs2 { void doit() { }}
- class Base2 { abstract class Abs3 extends Base2 { }}
- **class Base3 { abstract int var1 = 89; }**

Una clase abstracta puede estar dentro de otra clase como miembro interno.

Una clase abstracta puede contener métodos concretos (implementados). El uso de abstract no requiere que todos los métodos en la clase sean abstractos.

Una clase anidada puede ser declarada como abstracta, y también puede extender a su clase externa (aunque esto crea una estructura complicada).

una variable no puede ser marcada como abstract. El modificador abstract solo se aplica a clases o métodos. Las variables son concretas y deben tener un valor definido si no son declaradas como final static.

38. What is the result?

```
public static void main(String[] args) {  
    System.out.println("Result: " + 2 + 3 + 5);  
    System.out.println("Result: " + 2 + 3 * 5);  
}
```

- Result: 10 Result: 30
- Result: 25 Result: 10
- **Result: 235 Result: 215**
- Result: 215 Result: 215
- Compilation fails.

39. What is the result?

```
public class MyStuff {  
    String name;  
    MyStuff (String n) { name = n; }  
    public static void main (String[] args) {  
        MyStuff m1 = new MyStuff ("guitar");  
        MyStuff m2 = new MyStuff ("tv");  
        System.out.println (m2.equals(m1));  
    }  
    public boolean equals (Object o) {  
        MyStuff m = (MyStuff) o;  
        if (m.name != null) { return true; }  
        return false;  
    }  
}
```

- The output is true and MyStuff fulfills the Object.equals() contract
- The output is false and MyStuff fulfills the Object.equals() contract
- **The output is true and MyStuff does NOT fulfill the Object.equals() contract.**
- The output is false and MyStuff does NOT fulfill the Object.equals() contract

. La implementación compara únicamente si el atributo name del objeto m no es null, devolviendo true en ese caso.

40. Which one is valid as a replacement for foo?

```
public static void main(String[] args) {  
    Boolean b1 = true;  
    Boolean b2 = false;  
    int i = 0;  
    while (foo) { }  
}
```

- b1.compareTo(b2)
- i = 1
- i == 2? -1:0
- foo.equals("bar")

41. What is the result?

```
interface Rideable {  
    String getGait();  
}  
  
public class Camel implements Rideable {  
    int weight = 2;  
    String getGait() {  
        return " mph, lope";  
    }  
    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);  
    }  
}
```

- 16 mph, lope
- 24 mph, lope.
- **Compilation fails**
- 27 mph, lope.

42. What is the result?

```
class X {  
    String str = "default";  
    X(String s) { str = s; }  
    void print() { System.out.println(str); }  
    public static void main(String[] args) { new X("hello").print(); }  
}
```

- Hello
- Default
- Compilation fails.
- The program prints nothing.
- An exception is thrown at run time.

43. What is the result?

```
public static void main(String[] args) {  
    int[] array = { 1, 2, 3, 4, 5 };  
    System.arraycopy(array, 2, array, 1, 2);  
    System.out.print(array[1]);  
    System.out.print(array[4]);  
}
```

- 14
- 15
- 24
- 25
- 34
- 35

44. What is the result?

```
public class Main {  
    public static void main(String[] args) {  
        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;  
  
        System.out.println("El valor de w es: " + w);  
    }  
}
```

- 0
- 30
- 60

45. What is the result?

```
public class DoWhile {  
    public static void main(String[] args) {  
        int ii = 2;  
        do {  
            System.out.println(ii);  
        } while (--ii);  
    }  
}
```

- 2 1 2
- 1 0
- null
- An infinite loop.
- **Compilation fails**

46. What changes will make this code compile?

```
class X {  
    X() { }  
    private void one() { }  
}  
public class Y extends X {  
    Y() {}  
    private void two() {  
        one();  
    }  
    public static void main(String[] args) {  
        new Y().two();  
    }  
}
```

El método one() es private, por lo que no es accesible desde ninguna otra clase, ni siquiera por las clases que la extienden, como Y.

- Adding the public modifier to the declaration of class X.
- Removing the Y() constructor.
- Removing the private modifier from the two() method.
- Adding the protected modifier to the X() constructor.
- **Changing the private modifier on the declaration of the one() method to protected.**

47. Which two declarations will compile?

```
14.    public static void main(String[] args) {  
15.        Int a, b, c = 0;  
16.        int a, b, c;  
17.        int g, int h, int i = 0;  
18.        int d, e, f;  
19.        Int k, l, m, = 0;  
20.    }
```

- Line 15.
- Line 16.
- Line 17.
- Line 18.
- Line 19
- Line 20

48. Which three methods, inserted individually at line, will correctly complete class Two(Choose three)?

```
10.    class One {  
11.        void foo() { }  
12.    }  
13.    class Two extends One {  
14.        //insert method here  
15.    }
```

- public void foo() { /* more code here */ }
- private void foo() { /* more code here */ }
- protected void foo() { /* more code here */ }
- int foo() { /* more code here */ }

49. 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) { }
```

- If conn.rollback(s1) is inserted, account will be incremented by 10.
- If conn.rollback(s1) is inserted, account will be incremented by 1010.
- If conn.rollback(s2) is inserted, account will be incremented by 100
- If conn.rollback(s2) is inserted, account will be incremented by 110.
- If conn.rollback(s2) is inserted, account will be incremented by 1110

50. Whats is the result?

```
public static void main(String[] args) {  
    System.out.println("Result:" + 3 + 5);  
    System.out.println("Result:" + (3 + 5));  
}
```

- Result: 8 Result: 8
- Result: 35 Result: 8
- Result: 8 Result: 35
- Result: 35 Result: 35

51. What is the result?

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

La variable theString tiene el valor "Hello World", que tiene 11 caracteres en total. Los índices válidos son de 0 a 10

- There is no output.
- d is output.
- **A StringIndexOutOfBoundsException is thrown at runtime.**
- An ArrayIndexOutOfBoundsException is thrown at runtime.
- A NullPointerException is thrown at runtime.
- A StringArrayIndexOutOfBoundsException is thrown at runtime.

52. What will make this code compile and run?

```
01. public class Simple {  
02.  
03.     public float price;  
04.     public static void main(String[] args) {  
05.  
06.         Simple price = new Simple();  
07.         price = 4;  
08.     }  
09. }
```

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

53. In the Java collections framework a Set is:

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

54. 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 { }
```

- sc: class.Object asc: class.AnotherSampleClass
- sc: class.SampleClass asc: class.AnotherSampleClass
- sc: class.AnotherSampleClass asc: class.SampleClass
- **sc: class.AnotherSampleClass asc: class.AnotherSampleClass**

55. Which declaration initializes a boolean variable?

- `boolean j = (1 < 5);`
- `boolean m = null;`
- `boolean h = 1;`
- `boolean k = 0;`

56. Which two will compile, and can be run successfully using the following command?
Java Fred1 Hello walls.

- `abstract class Fred1 {public static void main(String[] args){System.out.println(args[2]);}}`
- `class Fred 1{ public static void main(String args) { System.out.println(args[1]); } }`
- `class Fred1 { public static void main(String[] args) { System.out.println(args[1]); } }`
- `class Fred1 { public static void main(String[] args) { System.out.println(args); } }`

57. 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;  
        }  
    }  
}
```

- Thrice
- It is not printed because compilation fails
- Twice
- Zero
- **Once**

58. Which two are valid instantiations and initializations of a multidimensional array?

- `array2D[0][0] = 1; array2D[0][1] = 2; array2D[1][0] = 3; array2D[1][1] = 4;`
- `array3D[0][0] = array; array3D[0][1] = array; array3D[1][0] = array; array3D[0][1] = array;`
- `int[][] array2D = {0, 1};`
- `int[][][] array3D = {{0, 1}, {2, 3}, {4, 5}}; int[] array = {0, 1}; int[][][] array3D = new int[2][2][2];`
- `int [][] array2D = {{0, 1, 2, 4}{5, 6}}; int[][] array2D = new int[2][2];`

59. The standard API for accessing databases in Java is:

- JPA/Hibernate JPA (Java Persistence API) es una especificación de Java para ORM (Object-Relational Mapping)
- **JDBC** DBC es la API estándar de Java para interactuar con bases de datos. Es parte de la biblioteca principal de Java y permite ejecutar sentencias SQL directamente contra bases de datos relacionales.
- ODBC es un estándar de acceso a bases de datos desarrollado por Microsoft.

60. What is the best way to test that the values of h1 and h2 are the same?

```
public static void main(String[] args) {  
    String h1 = "Bob";  
    String h2 = new String("Bob");  
}
```

- `if (h1.equals(h2)).`
- `if (h1 == h2).`
- `if (h1.toString() == h2.toString()).`
- `if (h1.same(h2)).`

61. What is the result?

```
1 public class Boxer1{
2     Integer i;
3     int x;
4     public Boxer1(int y){
5         x = i+y;
6         System.out.println(x);
7     }
8     public static void main(String[] args){
9         new Boxer1(new Integer(4));
10    }
11 }
```

- A `NullPointerException` occurs at runtime.
- The value "4" is printed at the command line.
- An `IllegalStateException` occurs at runtime.
- Compilation fails because of an error in line 9.
- Compilation fails because of an error in line 5.
- A `NumberFormatException` occurs at runtime.

62. Given:

```
We have the following Java class:
package gal.dicoruna.example;
public class C {
    protected String v;
    ...
}
```

- From the class, the package, subclasses.
- Any site but only read if it is outside the package.
- From the class, the package, subclasses and all sites.

63. What is the result?

```
class Foo{
    public void addFive(){ a += 5; System.out.println("f");}
}
class Bar extends Foo{
    public int a = 8;
    public void addFive(){ this.a += 5; System.out.println("b");}
}
Invoked with:
    Foo f = new Bar();
    f.addFive();
    System.out.println(f.a);
```

- b 3
- **Compilation fails**
- f 8
- b 8
- f 13
- f 3
- An exception is thrown at runtime. b 13

64. What is the result?

```
public class ScopeTest{
    int z;
    public static void main(String[] args){
        ScopeTest myScope = new ScopeTest();
        int z = 6;
        System.out.println(z);
        myScope.doStuff();
        System.out.print(z);
        System.out.print(myScope.z);
    }
    void doStuff(){
        int z = 5;
        doStuff2();
        System.out.print(myScope.z);
    }
    void doStuff2(){
        z = 4;
    }
}
```

- 6554
- 6564
- 6565
- **Compilation fails**
- 6566

65. What is the result?

```
public class Barn{
    public static void main(String[] args){
        new Barn().go("hi", 1);
        new Barn().go("hi", "world", 2);
    }
    public void go(String... y, int x){
        System.out.print(y[y.length - 1] + " ");
    }
}
```

- worl world.
- hi hi
- hi world
- An exception is thrown at runtime.
- **Compilation fails**

66. What is the result if you try to compile Truthy.java and then run it with assertions enabled?

```
public class Truthy{
    public static void main(String[] args){
        int x = 7;
        assert (x == 6) ? "x == 6" : "x != 6";
    }
}
```

- Truthy.java compiles and the output is x != 6
- Truthy.java compiles and an AssertionError is thrown with x != 6 as additional output.
- **Truthy.java does NOT compile.**
- Truthy.java compiles and an AssertionError is thrown with no additional output

67. What is the result? Given:

- 0 tom 0 jerry 1 tom 1 jerry 2 tom 2 jerry
- **0 tom 0 jerry 1 tom 2 tom 2 jerry**
- 0 tom 0 jerry 2 tom 2 jerry
- 0 tom 0 jerry 1 tom 1 jerry

68. What is the result when this program is executed?

```
public class Student{
    public String name = "";
    public int age = 0;
    public String major = "Undeclared";
    public boolean fulltime = true;
    public void display(){
        System.out.println("Name:" + name + "Major:" + major);
    }
    public boolean isFullTime(){
        return fulltime;
    }
}

And:
class TestStudent {
    public static void main(String[] args){
        Student bob = new Student();
        Student jian = new Student();
        bob.name = "Bob";
        bob.age = 19;
        jian = bob;
        jian.name = "Jian";
        System.out.println("Bob's Name:" + bob.name);
    }
}
```

- Nothing prints
- Bob's name.
- Bob's Name: Bob.
- **Bob's Name: Jian**

69. Which class has a default constructor?

```
class X{}  
class Y{  
    Y(){}  
}  
class Z{  
    Z(int i){}  
}
```

- Z only.
- X only.
- X, Y and Z
- **X and Y**
- X and Z
- Y only
- Y and Z

70. Which two may precede the word "class" in a class declaration?

- **Public**
- **Static**
- Synchronized
- Local
- Volatile

71. The BUILDER pattern is used to:

- Having several constructor methods in a class.
- **Simplify the creation of complex objects.**
- Implement the constructor of a class.

72. What is the result?

```
public class Main {  
    public static void main(String[] args) {  
        int a = 0;  
        a++;  
        System.out.println(a++);  
        System.out.println(a);  
    }  
}
```

- 0 1
- **1 2**
- 2 2
- 1 1

73. What value should replace kk in the comment to cause jj = 5 to be output?

```
public class MyFive {  
    public static void main(String[] args) {  
        short kk = 11;  
        short ii;  
        short jj = 0;  
        for (ii = kk; ii > 6; ii -= 1) {  
            jj++;  
        }  
        System.out.println("jj=" + jj);  
    }  
}
```

- -1
- 1
- 5
- 8
- **11**

74. What is a static block of code in Java?

- A block of code within a class that runs whenever the class is load on the JVM.
- **A block of code inside a class that runs when that class first loaded in JVM**
- A block of code within a class that always runs before the builder.

Un bloque estático en Java es un segmento de código dentro de una clase que se define con la palabra clave static. Este bloque se ejecuta una única vez, la primera vez que la clase se carga en la JVM, antes de que se cree cualquier instancia de la clase o se acceda a cualquier miembro estático.

75. What is the result?

```
class X {
    X() {
        System.out.print(1);
    }
    X(int x) {
        this();
        System.out.print(2);
    }
}

public class Y extends X {
    Y() {
        super(6);
        System.out.print(3);
    }
    Y(int y) {
        this();
        System.out.print(4);
    }

    public static void main(String[] a) {
        new Y(5);
    }
}
```

- 2134
- 4321
- 2143
- 13
- **1213**
- 134

76. Which two actions, used independently, will permit this class to compile?

```
import java.io.IOException;

public class Y {
    public static void main(String[] args) {
        try {
            doSomething();
        } catch (RuntimeException e) {
            System.out.println(e);
        }
    }

    static void doSomething() {
        if (Math.random() > 0.5) {
            throw new IOException();
        }
        throw new RuntimeException();
    }
}
```

- Adding throws IOException to the main() methodIOException
- Adding throws IOException to the dosomething() method signature and changing the catch
- Adding throws IOException to the main() method signature and to the dosomething() method
- Adding throws IOException to the main() method signature
- Adding throws IOException to the doSomething() method signature.

77. A method is declared to take three arguments. A program calls this method and passes only two arguments. What is the result?

- Compilation fails
- The third argument is given the value void.
- The third argument is given the value zero
- An exception occurs when the method attempts to access the third argument. The third argument is given the appropriate false value for its declared type. The third argument is given the value null.

cuando declaras un método que requiere un número específico de argumentos, la llamada al método debe proporcionar exactamente esa cantidad de argumentos. Si faltan argumentos o se proporcionan más de los esperados, el programa no compila.

78. What is the result?

```
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;
    }
}
```

- 6565
- 6504
- 6560
- 6550