Guía JAVA

1. Excepciones de Java permitida*

- org.springframework.web.client.RestClientException
- No conozco la respuesta
- java.lang.NumberFormatException
- com.bbva.apx.exception.db.NoResultException
- com.bbva.elara.utility.interbackend.cics.exceptions.BusinessException

2. Which three are bad practices?

- Checking for ArrayIndex0ut0fBoundsException 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 ArrayIndex0ut0fBoundsException 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
- 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 statments 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 independiently 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.

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 excuted?

```
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");
        }
}
```

- 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();
        }
}
```

- 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());
        }
}
```

- 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);
        }
}
```

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

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

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.

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.I0Exception { // 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.I0Exception { // do something } }
- class Test implements SampleCloseable { public void close() { // do something }}

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.tellItLikeItIs();
18.          (Truth) speakIT.tellItLikeItIs();
19.          ((Truth) speakIT).tellItLikeItIs();
20.          tellIt.tellItLikeItIs();
21.          (Truth) tellIt.tellItLikeItIs();
22.          ((Truth) tellIt).tellItLikeItIs();
23.      }
24.  }

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

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

- 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.

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 = (0bject) 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 ArrayIndex0ut0fBoundsException is thrown at run time.

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 "
id; }
19. }
20.
21. public class EmployeeTest {
22.  public static void main(String[] args)
{
23.   Employee e = new Employee("4321");
24.   System.out.println(e.empID);
25.  }
26. }
```

- 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; }

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

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?

```java
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();
        }
}
```

- 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 declarationof 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));
        }
}
```

- There is no output.
- d is output.
- A StringIndex0ut0fBoundsException is thrown at runtime.
- An ArrayIndex0ut0fBoundsException is thrown at runtime.
- A NullPointException is thrown at runtime.
- A StringArrayIndex0ut0fBoundsException 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.0bject 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 usaing 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.printIn(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 = newint[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
- JDBC
- ODBC

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

```
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 additionaloutput.
- Truthy.java does NOT compile.
- Truthy.java compiles and an AssertionError is thrown with no additional output

67. What is the result? Given:

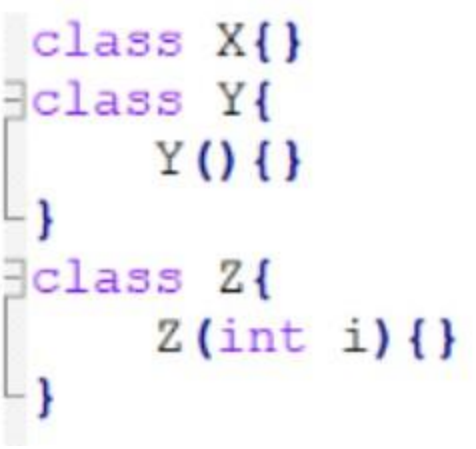
- 0 tom 0 jerry 1 tom 1 jerry 2 tom 2 jerry
- 0 tom 0 jerry 1 tom 2 tom 2 jerry
- 0 tom 0 jerry 2 tom 2 jerry
- 0 tom 0 jerry 1 tom 1 jerry

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.

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?

```java
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() method ....IOException
- 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 doSoomething() method signature.

77. A method is declared to take three arguments. A program calls this method andpasses 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.

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

Cuestionario 2

1. ¿Qué es una API?
   interfaz de programación de aplicaciones

2. ¿Qué es REST?
JavaScript Object Notation

3. ¿Que es un verbo HTTP?
HTTP define un conjunto de métodos o verbos de petición para indicar la acción que se desea realizar para un recurso determinado

4. Menciona al menos 3 verbos en HTTP
GET, POST, PUT

5. ¿En términos REST que significa idempotencia?
La ejecución repetida de una petición con los mismos parámetros sobre un mismo recurso tendrá el mismo efecto en el estado de nuestro recurso en el sistema si se ejecuta 1 o N veces

6. Menciona cuales son los verbos idempotentes
GET, PUT, DELETE

7. Menciona que verbo no es idempotente.
POST

8. Un verbo POST se utiliza para eliminar un recurso
Falso

9. Un verbo PATCH no se utiliza para actualizar un recurso de manera parcial.
Falso

10. Verbo HTTP que se utiliza para obtener/listar información
GET

11. ¿Qué es un código de retorno HTTP?
El código HTTP es en el que se indica si se ha completado satisfactoriamente o no una solicitud HTTP específica.

12. ¿Qué códigos de retorno HTTP se clasifican como informativos?
100

13. ¿Qué códigos de retorno HTTP se clasifican como de redireccionamiento?
300

14. ¿Qué códigos de retorno HTTP se clasifican como de respuesta satisfactoria?
200

15. ¿Qué códigos de retorno HTTP se clasifican como error de cliente?
400

16. ¿Qué códigos de retorno HTTP se clasifican como error del servidor?
500

17. Un código de error 400 se utiliza cuando hay un error en el servidor
Verdadero
Falso

18. Un código 201 se utiliza para devolver una respuesta satisfactoria pero con un contenido vacío.
Verdadero

19. ¿A qué le llamamos endpoint en un servicio REST?
Son las URLs de un API o un backend que responden a una petición.

20. ¿Cuál es la estructura correcta de un endpoint?
GET apiRest/v0/products}

21. ¿Qué es un Query Param en un servicio REST?
Son parámetros que son enviados al final del endpoint y siempre deberán estar después de un signo de interrogación.

22. ¿Qué es un UriParam en un servicio REST?
Parámetro de ruta, se usa básicamente para identificar un recurso o recursos específicos.

ASO

1. Comando Git para mostrar el historial de commits
   git log

2. Comando GIT para crear un rama
   git checkout -b

3. Comando UNIX/Linux para borrar un archivo
   rm

4. Comando para iniciar un contenedor en Docker
   docker start

5. Comando de UNIX/Linux para mostrar la ruta actual
   pwd

6. Comando UNIX/Linux para crear una carpeta
   mkdir

7. Selecciona el comando Git correcto según la definición: "Crea un nuevo commit que deshace todos los cambios hechos en el commit, luego los aplica a la rama actual"
   git revert

8. Comando de UNIX/Linux para mover un archivo
   mv

9. Comando de UNIX/Linux para copiar un archivo
   cp

10. Comando para eliminar un contenedor en Docker
    docker rm

11. Comando Git para crear un repositorio
    git init

12. Comando para detener un contenedor en Docker
    docker stop

13. Comando UNIX/Linux para listar el contenido de una carpeta
    ls

14. Comando UNIX/Linux para moverte entre los directorios
    cd

15. Comando Git para agregar archivos al Staging
git add

16. Comando para reescribir el historial de Git
git commit -amend

HTTP—----------

17. Que significa el código 400: «Bad Request»

18. El verbo POST:
Se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

19. Que significa el código 200: «OK»

20. WSDL: Web Services Description Language

21. Que significa el código 204: «No Content»

22. El verbo PUT: reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

23. El verbo PATCH: Es utilizado para aplicar modificaciones parciales a un recurso.

24. El verbo GET: solicita una representación de un recurso específico

25. Que significa el código 500: «Internal Server Error»

26. Que significa el código 404: «Not Found»

27. Que es idempotencia: Es la propiedad de realizar una acción varias veces y conseguir siempre el mismo resultado

28. DELETE: borra un recurso en específico.

29. WADL: Web Application Description Language.

30. Cual de los siguientes verbos HTTP no es idempotente: POST

31. 504 : «Gateway Timeout»

Conceptos y Spring

32. Nombra los métodos de inyección de dependencias

- Inyección vía constructor
-Inyección vía métodos
-Inyección vía propiedades

33. Maven permite:

-Creación de documentación

-Creación de informes
-Resolución de dependencias
-Releases Distribuciones
-Compilaciones
-Todas las anteriores

34. POM: Project Object Model, es un fichero XML, que es la unidad principal de un proyecto Maven

35. Selecciona los puntos correctos sobre: API (Selecciona 3):

- significa Application Programming Interface
- permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados
- simplifica el desarrollo de aplicaciones

36. Cual de las siguientes afirmaciones es correcta sobre la inyección de dependencias:

Es un patrón de desarrollo de software donde los objetos no son responsables de inicializar sus dependencias

37. Nombra los 3 estereotipos básicos de Spring

@Repository
@Service
@Controller

38. Selecciona los puntos correctos sobre: JSON (Selecciona 3)

es un formato ligero de intercambio de datos
significa JavaScript Object Notation
para las máquinas es simple interpretarlo y generarlo

39. Que descripción se adapta mejor a la definición del Contenedor de Spring?

Se encarga de crear los objetos, conectarlos entre sí, configurarlos y además controla los ciclos de vida de cada uno objeto mediante el patrón de inyección de dependencias

40. Notación para quitar los warnings

@SuppressWarnings

41. Nombra los 3 scopes básicos de un Bean

Singleton
Singleton
Request

42. Selecciona la definición de scope SESSION

El contenedor de Spring creará una nueva instancia del objeto definido por el Bean para cada una de las sesiones HTTP (Aplicación Stateful) y entregará esa misma instancia cada vez que reciba una petición dentro de la misma sesión.

Información adicional

Funciones Principales de Postman

Envío de Solicitudes HTTP:

- GET: Solicitar datos de un servidor.
- POST: Enviar datos para crear un nuevo recurso.
- PUT: Actualizar un recurso existente.
- DELETE: Eliminar un recurso.

El verbo HTTP PATCH se utiliza precisamente para actualizar un recurso de manera parcial. A diferencia de PUT, que generalmente se utiliza para actualizar un recurso completo, PATCH permite modificar sólo una parte del recurso. Esto es útil cuando solo necesitas cambiar ciertos campos de un recurso sin tener que enviar toda la representación del recurso.

Diferencias Entre PATCH y PUT

- PATCH: Se usa para aplicar cambios parciales a un recurso. Sólo necesitas enviar los datos que deseas modificar.
  - Ventajas: Menor carga de datos, especialmente útil para grandes recursos.
  - Ejemplo: Cambiar solo el campo email de un usuario.
- PUT: Se usa para actualizar completamente un recurso. Generalmente, envías toda la representación del recurso, incluso los campos que no se modifican.
  - Ventajas: Facilita la sustitución completa del recurso.
  - Ejemplo: Actualizar todo el objeto usuario.

Categorías de Códigos de Estado HTTP

Los códigos de estado HTTP están organizados en cinco categorías, cada una representada por el primer dígito del código de tres cifras:

1. 1xx (Informativos):
   - Indican que la solicitud ha sido recibida y el proceso está en curso.
   - 100 Continue: El servidor ha recibido los encabezados de la solicitud y el cliente debe continuar enviando el cuerpo de la solicitud (si lo hay).

2. 2xx (Éxito):
   - Indican que la solicitud se ha recibido, entendido y procesado con éxito.
   - 200 OK: La solicitud ha sido exitosa.
   - 201 Created: La solicitud ha sido exitosa y ha resultado en la creación de un nuevo recurso.
   - 204 No Content: La solicitud ha sido exitosa, pero no hay contenido que devolver.

3. 3xx (Redirección):
    - Indican que se necesita tomar una acción adicional por parte del cliente para completar la solicitud.
    - 301 Moved Permanently: El recurso solicitado ha sido movido permanentemente a una nueva URL.
    - 302 Found: El recurso solicitado está temporalmente en una ubicación diferente, pero el cliente debe usar la URL original para futuras solicitudes.
    - 304 Not Modified: El recurso no ha sido modificado desde la última solicitud.

4. 4xx (Errores del Cliente):
    - Indican que la solicitud contiene errores que impiden que el servidor la procese.
    - 400 Bad Request: La solicitud está mal formada o es ilegible.
    - 401 Unauthorized: Se requiere autenticación para acceder al recurso.
    - 403 Forbidden: El servidor entiende la solicitud, pero se niega a autorizarla.
    - 404 Not Found: El recurso solicitado no se ha encontrado en el servidor.

5. 5xx (Errores del Servidor):
    - Indican que el servidor ha encontrado una condición que le impide completar la solicitud.
    - 500 Internal Server Error: Ha ocurrido un error genérico en el servidor.
    - 502 Bad Gateway: El servidor, actuando como puerta de enlace o proxy, ha recibido una respuesta no válida del servidor de respaldo.
    - 503 Service Unavailable: El servidor no está disponible temporalmente, generalmente debido a mantenimiento o sobrecarga.

¿Que es idempotencia?

En un sentido general, se refiere a una propiedad que posee una operación o función, la cual, cuando se aplica múltiples veces al mismo elemento o conjunto de elementos, produce el mismo resultado que si se aplicara una sola vez.

En el contexto de la informática y las API (Interfaces de Programación de Aplicaciones), la idempotencia es una propiedad deseable para ciertas operaciones. Cuando una operación es idempotente, realizarla una vez o varias veces tiene el mismo efecto; es decir, el resultado final es el mismo.

La idempotencia es una propiedad que asegura que aplicar una operación una o múltiples veces no altera el resultado final más allá de la primera aplicación de dicha operación. Esto es crucial en sistemas distribuidos y en el diseño de APIs para garantizar la integridad y consistencia de los datos.

Definición del Contenedor de Spring

El contenedor de Spring, también conocido como contenedor IoC (Inversión de Control) o contenedor de aplicaciones, es uno de los aspectos fundamentales del marco de trabajo Spring. Proporciona un entorno de ejecución para las aplicaciones de Spring, gestionando la creación, configuración y administración de los objetos (beans) de la aplicación.
Aquí tienes una definición detallada del contenedor de Spring:

1. Gestión de Objetos (Beans): El contenedor de Spring es responsable de crear y gestionar los objetos de la aplicación, conocidos como beans. Estos beans pueden ser objetos de negocio, componentes de acceso a datos, servicios web, o cualquier otro componente de la aplicación.
2. Inyección de Dependencias: El contenedor de Spring aplica el patrón de Inyección de Dependencias (DI), lo que significa que se encarga de proveer las dependencias requeridas por un bean cuando éste es creado. Esto promueve la modularidad y la desacoplamiento entre los componentes de la aplicación.
3. Configuración Declarativa: Spring permite configurar los beans de la aplicación de forma declarativa, utilizando XML, anotaciones o código Java. Esta configuración se utiliza para definir los beans, sus dependencias y cómo se deben inicializar.
4. Gestión del Ciclo de Vida de los Beans: El contenedor de Spring controla el ciclo de vida de los beans, inicializándolos cuando son necesarios y liberando los recursos asociados cuando ya no son necesarios. Esto garantiza una gestión eficiente de los recursos y la eliminación de posibles problemas de manejo de memoria.
5. Aspectos y Transacciones: El contenedor de Spring también puede administrar aspectos, que son aspectos de la funcionalidad de la aplicación que se aplican de manera transparente, como la gestión de transacciones. Esto simplifica la implementación de funcionalidades comunes y mejora la modularidad de la aplicación.

En resumen, el contenedor de Spring es el núcleo del marco de trabajo Spring, proporcionando un entorno de ejecución flexible y poderoso para la construcción de aplicaciones empresariales robustas y escalables. Su capacidad para gestionar objetos, inyectar dependencias, configurar declarativamente y administrar el ciclo de vida de los beans es fundamental para el desarrollo efectivo de aplicaciones con Spring.

Métodos de inyección de dependencias

Más comunes en Spring son:

1. **Inyección por Constructor**: En este método, las dependencias requeridas por una clase se pasan al constructor de la clase. Spring utiliza el constructor de la clase para crear una instancia y proporcionar las dependencias necesarias.

2. **Inyección por Setter**: En este método, las dependencias requeridas se proporcionan a través de métodos de "setter" en la clase. Spring llama a estos métodos de setter después de crear una instancia de la clase y antes de devolverla como un bean.

3. **Inyección por Campo (Field Injection)**: En este método, las dependencias requeridas se anotan directamente en los campos de la clase. Spring utiliza reflexión para asignar valores a estos campos después de crear una instancia de la clase.

Cada uno de estos métodos tiene sus propias ventajas y desventajas, y su elección depende de las necesidades específicas del proyecto y las preferencias de diseño del desarrollador. Sin embargo, el método preferido en Spring es la inyección por constructor debido a que promueve una mejor inmutabilidad y hace que las dependencias requeridas sean explícitas en la creación del objeto.