Rodrigo Martínez Zambrano

1.

Which three are bad practices?

*

1/1

Checking for an IOException and ensuring that the program can recover if one occurs.

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.

Para este caso, verificaremos todas las opciones:

a. “Comprobar si hay una IOException y asegurarse de que el programa pueda recuperarse si se produce una”.
   Se trata de una buena práctica. El manejo de IOException permite que un programa se ocupe de errores de entrada/salida de forma controlada.

b. “Comprobar si hay ArrayIndexOutOfBoundsException y asegurarse de que el programa pueda recuperarse si se produce una”.
   Esta es una mala práctica. ArrayIndexOutOfBoundsException normalmente indica un error de programación (por ejemplo, límites de bucle incorrectos). En lugar de detectar esta excepción, se debe corregir el código para evitar que se produzca.

c. “Comprobar si hay FileNotFoundException para informar a un usuario de que un nombre de archivo introducido no es válido”.
   Se trata de una buena práctica. Permite que el programa proporcione un mensaje claro al usuario y gestione el error de forma elegante.

d. “Comprobar si hay errores y, si es necesario, reiniciar el programa para asegurarse de que los usuarios no sepan que hay problemas”.
   Esta es una mala práctica. Los errores representan problemas graves que un programa no debería intentar solucionar, como OutOfMemoryError o StackOverflowError. Por lo general, no se pueden recuperar y no se deben detectar para intentar recuperarlos.

e. “Comprobación de ArrayIndexOutOfBoundsException al iterar a través de una matriz para determinar cuándo se han visitado todos los elementos”.
   Esta es una mala práctica. En lugar de usar excepciones para el flujo de control, el bucle debería estar correctamente delimitado para evitar acceder a índices no válidos.

2.

```
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?
```

El arreglo está inicializado y es contiene dos filas, una de ellas [0] de tamaño 3 y [1] de tamaño 4.

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

array2D[0].length = 3, corresponde a la longitud e la primer fila del arreglo.

array2D[1].length.getClass().IsArray() = true, esto resulta true ya que estos métodos se utilizan para obtener la clase y comprobar si es un arreglo respectivamente, al declarar array2D como un arreglo, esto imprimirá true.

array2D[0][1] = 1, esto corresponde al valor de la primer fila y la segunda columna.

3.

Which two statments are true?

*

1/1

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.

Estos enunciados corresponden al concepto de herencia, si es posible heredar una clase de una clase abstracta y también es posible que una clase abstracta implemente una interfaz. Al hacer esto, se deberán implementar los métodos correspondientes para que el programa compile correctamente.

4.

```
Given:

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 = (Gamma) new Alpha();
        Gamma g2 = (Gamma) new Beta();
        System.out.print(g1.getType()+ " " +g2.getType());
    }
}
What is the result?
```

La jerarquía de clases es de la siguiente forma:

La ejecución de este código va a resultar en un error de compilación. Esto se debe a esta parte de código:

```
Gamma g1 = (Gamma) new Alpha();
Gamma g2 = (Gamma) new Beta();
```

No se puede convertir una instancia de una superclase (Alpha o Beta) a una subclase (Gamma) porque la superclase no contiene la estructura y el comportamiento adicionales de la subclase. La subclase puede tener campos y métodos adicionales que no existen en la superclase, lo que hace que dicha conversión no sea válida y genere una ClassCastException en tiempo de ejecución.

5.

Which five methods, inserted independiently at line 5, will compile? (Choose five)
*

1/1

```
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; }

En el caso de los métodos de la imagen:

Private int blipvert(int x) { return 0; }
Protected long blipvert(int x) { return 0; }

El primero provocaría error de compilación debido a que al hacer Override a un método, no es posible disminuir su visibilidad (cambia de protected a private).

En el segundo, no es posible cambiar el tipo de retorno del método (regresa long en lugar de int).

6.

```
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);}

Respuesta correcta

Change line 13 to: public Sub(){ super(5);}
Change line 13 to: public Sub(){ this(5);}

En este caso,la línea 13 genera un error de compilación ya que no podrá ejecutar this.a debido a que a es un atributo privado en la clase Super.

Para solucionar esto, las respuestas correctas serían:

b. Change line 13 to: public Sub(){super(5);} esto mandaría llamar al constructor de la clase Super y establecería a = 5.

d. Chance line 13 to: public Sub(){this(5);}, esta modificación haría que this mande llamar al otro constructor que recibe de parámetro un entero y este a su vez manda llamar al constructor de Super.

7. 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.

No hay nada de malo en el código, se define una clase abstracta que a su vez define atributos y métodos que podrán ser implementados por otras clases que hereden de ésta. Si acepta variables de instancia, métodos privados, no son necesarios los constructores sin argumentos ni tampoco es necesario tener al menos un método abstracto.

8.

What is the result?
*
1/1

```
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(); }
}
```

Atom granite atom granite.

Se puede explicar por pasos:

1. El método main crea una instancia de Mountain.
2. Mountain ejecuta super(“granite”), esto llama al constructor de su clase padre (Rock) que recibe un string.
3. Rock antes de ejecutar su print, manda llamar al constructor de su clase padre (Atom) el cual imprimirá “Atom”.
4. Ahora Rock imprimirá “granite”.

5. El constructor de Mountain creará una instancia de Rock y se repiten los pasos 3 y 4.
6. LA ejecución del programa entonces dará como resultado “atom granite atom granite”

9.

What is printed out when the program is excuted?
*

1/1

```
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

Se ejecutará el método main que empiece por public y sea estático, en este caso es posible agregar final al método principal de un programa. El resultado será “three”.

10. 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 c.
Feline cougar c f.

El resultado es “feline cougar c c” esto se debe a:

1. En el método main, se instancia un objeto de Cougar, al ser una clase hija, implícitamente su contructor manda llamar al constructor de la clase padre (Feline).
2. La clase Feline inicializa la variable type a “f” e imprime “feline”
3. Cougar imprime cougar”.
4. Se ejecuta el método go() que le asigna el valor de c a type, posteriormente imprime this.type que hace referencia a la variable de la clase y super.type se refiere a la misma variable, de la clasa padre, resultando en “c c”.

11.

What is the result?
*
1/1

```
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);
        }
}
```

La ejecución del programa arrojaría una excepción, ya que Scanner leería enteros y “a” al no ser de tipo entero, lanzaría InputMismatchException, la excepción se lanza cuando un método de entrada de un Scanner no coincide con el tipo de datos que se espera.

12.

What is the result?
*
1/1

```
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.

La línea t1.wait(5000); lanzará una excepción IllegalMonitorStateException. Como resultado, el bloque try-catch en main() capturará esta excepción y se ejecutará la instrucción en el bloque catch.

El resultado será “1 thrown to main”.

13. Wich 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.

El resultado de ejecutar el programa será 420, no hay ningún problema con el código y se ejecutará como se indica.

14.

The SINGLETON pattern allows:

*

0/1

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.

La respuesta correcta sería “Having a single instance of a class, while allowing all clases have Access to that instance”. El patrón Singleton se encarga de crear una única instancia para una clase en específico y que todos los objetos creados se referencien a esa instancia, para evitar que si se crea más de un objeto se realice más de una instancia,

15. 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.

1. **Inicialización del arreglo de cadenas sa:** Se inicializa el arreglo sa con dos valores: "111.234" y "222.5678".
2. **Obtención de una instancia de NumberFormat:** Se obtiene una instancia de NumberFormat mediante NumberFormat.getInstance().
3. **Configuración del formato numérico:** Se establece el número máximo de dígitos fraccionarios a 3 mediante nf.setMaximumFractionDigits(3).
4. **Iteración sobre el arreglo sa:** Se itera sobre cada cadena en el arreglo sa y se intenta parsear cada cadena como un número utilizando nf.parse(s).
5. El primer número, "111.234", se parseará sin problemas y se imprimirá como 111.234.
6. El segundo número, "222.5678", se parseará sin truncamiento ni redondeo y se imprimirá como 222.5678.

16.

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 ();

La respuesta correcta sería:

Square square = new Square();square.foo();square.foo(“bar”);

1. Al crear el objeto Square(), se llama a su constructor que a su vez llama al contructor de la clase padre (Shape) el cual imprimirá “Shape: constructor”.
2. Square.foo(), llamará al método foo() de la clase padre, el cual imprimirá “Shape: foo”.
3. Square.foo(“shape”) al recibir un parámetro, llamará a su propio método foo(String label), por lo que se imprimirá “quare: foo”.

17. Wich 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 }}

Respuesta correcta

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

La primera opción es una implementación correcta ya que lanza la misma excepción que la interfaz.

La tercera opción también sería una opción válida ya que FileNotFoundException pertenece a IOException al ser un error de entrada.

La última opción también es válida ya que se puede implementar sin especificar la excepción.

18. 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());
```

El resultado sería 4. El método put de un objeto tipo HashMap agrega los valores indicados: el HashMap sería:

[1,car],[2,boat],[1,plane],[2,bus]

El cual tiene un tamaño de 4. El método equals está sobrescrito para comparar las claves basándose en el valor del campo key. Esto significa que m1 y m3 son considerados iguales (ambos tienen key = 1), y m2 y m4 son considerados iguales (ambos tienen key = 2).

Por defecto, el método `hashCode` de `Object` devuelve un valor diferente para cada instancia, incluso si los objetos son "iguales" según el método `equals`.

19. 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;

Para imprimir la secuencia solicitada, los valores deben ser x= 2, y=3, z=4

- El bucle for se ejecuta 2 veces (x=2).
- Dentro de cada iteración del bucle for, el bucle do-while se ejecuta 3 veces (ya que y = 3).

- Dentro de cada iteración del bucle do-while, el bucle while se ejecuta 4 veces (ya que z = 4), esto hará que se imprima 1234.
- Después de completar las iteraciones del bucle do-while, se imprime --- como separador.

20. 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();
}
```

21. ¿Que acción realiza el archivo de dependencia pom.xml?

El archivo pom.xml recupera todas las dependencias con otros proyectos.

22.

¿Que causa no corresponde al antipatron Contenedor Magico?*

1/1

Reparto de responsabilidades incorrecto
Falta de complimiento de la arquitectura
Intervención demasiado limitada
Desacoplamiento entre librerias

El anti patrón "Contenedor Mágico" generalmente se refiere a situaciones donde se abusa de contenedores o estructuras de datos avanzadas de manera incorrecta, resultando en un código difícil de mantener y depurar.

El desacoplamiento entre librerías es un concepto positivo en diseño de software que se refiere a minimizar las dependencias entre diferentes módulos o librerías, lo cual facilita el mantenimiento, la prueba y la reutilización del código. Este concepto no está relacionado directamente con los problemas causados por el uso de contenedores mágicos.

Por lo tanto, "Desacoplamiento entre librerías" no corresponde al anti patrón "Contenedor Mágico".

23.

El objeto _____ no debe manejarse directamente. No debe crear instancias , abrir o cerrar explicitamente conexiones a la base de datos*

1/1

Datasource

JPA
JDBC
Database

El objeto **Datasource** no debe manejarse directamente. No debe crear instancias, abrir o cerrar explícitamente conexiones a la base de datos.

Datasource es un objeto de Java que se utiliza para manejar conexiones a bases de datos. Su uso es preferido sobre la creación manual de conexiones mediante DriverManager

24.

En un Batch, ¿Cual es el step donde el desarrollador introducirá la logica empresarial necesaria?*

0/1

itemWriter

Task
Chunk
JobLauncher

Respuesta correcta

Task

Task en el contexto de procesamiento por lotes (Batch Processing) se refiere a una unidad de trabajo que puede incluir la lógica empresarial necesaria para transformar, procesar o analizar los datos.

25.

El identificador unico para la ejecución de transacciones/Job se genera automaticamente como ultimo paso cuando se invoca las transacciones en linea/Batch. El resultado de la transaccion se puede propagar para la correlación en las otras capas.*

1/1

Falso
Verdadero

En el contexto de procesamiento de transacciones o ejecución de trabajos por lotes (Batch Jobs), es común que se genere un identificador único para cada ejecución. Este identificador único es crucial para varias razones:

1. **Seguimiento y Registro**: Permite rastrear y registrar cada ejecución de transacción o trabajo por lotes de manera única.
2. **Correlación de Resultados**: El identificador puede propagarse a través de diferentes capas del sistema, permitiendo correlacionar los resultados de la transacción con otras operaciones relacionadas en diversas capas de la arquitectura.
3. **Gestión de Errores**: Facilita la identificación y manejo de errores específicos asociados a una ejecución particular.
4. **Auditoría y Monitoreo**: Proporciona un mecanismo para auditar y monitorear las ejecuciones de trabajos y transacciones.

Por lo que la afirmación anterior resulta verdadera.

26.

Un _____ proporciona una solución a un problema de diseño. Debe cumplir con diferentes características, como la efectividad al resolver problemas similares en ocasiones anteriores. Por lo tanto, debe de ser re-utilizable, es decir, aplicable a diferentes problemas en diferentes circunstancias.

Patrón de diseño, existen múltiples patrones de diseño y cada uno realiza una tarea específica, se debe elegir bien con cuál se trabajará e incluso se pueden combinar, esto permitirá una arquitectura limpia y hará el código reutilizable y escalable.

27. What is the result?

```
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);
        }
}
```

El resultado será 7 5 1. Ya que el método remove(int index) borrará del ArrayList el objeto que se encuentre en el índice específicado, en este caso corresponde a la posición 1 que es el valor 1.

Valor	7	1	5	1
índice	0	1	2	3

28.

Para enviar un aviso desde una biblioteca o desde la transacción, cualquiera que sea el tipo, el método utilizado es _____.

addSeverity
addError
addAdvice
addWarning

La respuesta correcta es AddAdvice, este método se utiliza para registrar mensajes de advertencia que pueden ser importantes para el monitoreo o la depuración de una aplicación.

29.

¿Quién es el responsable de iniciar un Job?
*

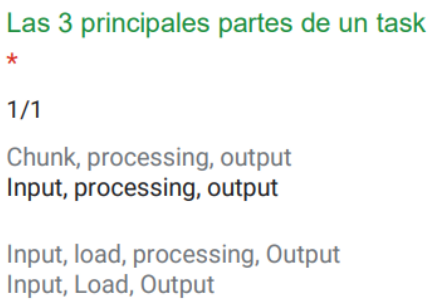
1/1

JobExecution
JobLauncher

JobInstance
JobBegginer

JobLauncher: Es responsable de lanzar y gestionar la ejecución de trabajos (Jobs) dentro de un proceso Batch.

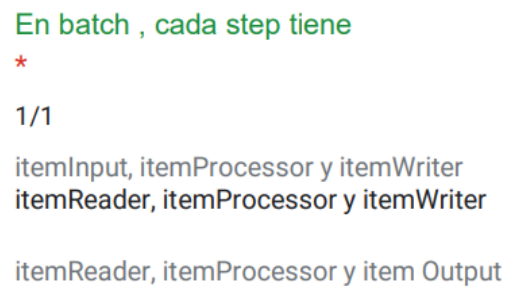
30.



Las tres principales partes de un "task" (tarea) en términos generales, especialmente en el contexto de procesamiento de datos y sistemas batch, son:

1. **Input** (Entrada): Es la fase donde se recopilan los datos que serán procesados.
2. **Processing** (Procesamiento): Es la fase donde los datos de entrada son procesados, transformados o analizados según la lógica del negocio.
3. **Output** (Salida): Es la fase donde los datos procesados son exportados, almacenados o enviados a un destino final.

31.



Los componentes de un step son:

- Item Reader: Lee los datos de entrada en fragmentos (chunks) desde una fuente de datos.
- Item Processor: Procesa o transforma cada fragmento de datos leído.
- Item Writer: Escribe los datos procesados a una salida, que puede ser una base de datos, un archivo, etc.

32.

¿Cuál es la ubicación de archivos temporales utilizados en Batch?
*

1/1

/fichtemporal/datent
/fichtemcomp/datent

/temporalFiles/datent
/datentJobs/datent

33.

¿Qué característica NO pertenece al BATCH?
*

1/1

Procesamiento generalmente grande volúmenes de información
Interacción con usuario (directo)

El Batch es un procesamiento por lotes automático, no está relacionada una interacción directa con el usuario.

34.

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.
*

1/1

Mockito
Mock

Inject
InjectMock

La anotación @Mock en el contexto de pruebas unitarias con JUnit y Mockito se utiliza para indicar que una propiedad (o campo) debe ser simulada (mocked). Esto significa que se crea una instancia simulada del objeto, lo cual permite controlar el comportamiento del objeto durante las pruebas, sin necesidad de depender de su implementación real.

35.

¿Cuál es el objetivo de trabajar con Patrones?
*

1/1

Los patrones de diseño sirven de apoyo para la búsqueda de soluciones a problemas comunes a la hora del desarrollo de software.
Brindar una solución a un problema de diseño. Para que una solución se pueda considerar un patrón debe cumplir con diferentes características, como la efectividad a la hora de resolver problemas similares en ocasiones anteriores. Por ende, debe ser reutilizable, es decir, aplicable a diferentes problemas en distintas circunstancias.

Estos componentes contienen información necesaria para la ejecución de la Transacción en APX y cualquier modificación al respecto podría alterar su comportamiento
a y b
a y c

Utilizar patrones de diseño se utiliza para dar solución a un problema de diseño, existen varios tipos de patrones y cada uno cumple una característica, sin embargo, todos ayudarán a que nuestro código sea más eficiente, escalabre y reutilizable.

36.

En tiempo de ejecución, las dependencias con otras bibliotecas se resuelven en función de los paquetes declarados en la sección...
*

1/1

Abstract
dependency
Test
No conozco la respuesta
import-package

Al especificar los paquetes en import-package, el framework puede gestionar dinámicamente las dependencias entre módulos, permitiendo una mejor modularidad y gestión de versiones en la aplicación.

37.

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

El resultado del código anterior será “2 2”.

1. b se inicializa en 4
2. b—es un post-decremento, por lo que la siguiente línea b = 3.
3. –b es un pre-decremento, por lo que se le restara 1 y se imprimirá 2.
4. Volverá a imprimir el valor de b, es 2.

38.

In Java the difference between throws and throw Is:*

1/1

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.

En Java, throw y throws se utilizan para manejar excepciones, pero tienen propósitos diferentes:
throw

- **Uso**: throw se utiliza dentro de un método para lanzar explícitamente una excepción.
- **Sintaxis**: throw new ExceptionType("Mensaje de error");

throws

- Uso: throws se utiliza en la declaración de un método para indicar que el método puede lanzar una o más excepciones.
- Sintaxis: public void nombreMetodo() throws ExceptionType1, ExceptionType2 { ... }

39. **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();

La respuesta correcta es sc=asc, asc es una instancia de AnotherSampleClass y AnotherSampleClass es una subclase de SampleClass, por lo que una instancia de AnotherSampleClass puede ser asignada a una referencia de tipo SampleClass debido a la herencia.

40.

What is the result?*

1/1

```
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.

El código inicialmente imprimirá un 4, que corresponde al valor de la posición [4][1], eso es, el índice 1 del sub arreglo 4.

Posteriormente se quiere imprimir el valor de la posición [1][4], sin embargo, el sub arreglo 1 solamente contiene 2 elementos por lo que se generará una excepción.

… HASTA LA PAG 127