Academia Java

Guia APX:

Java

Capacitador:

Miguel Angel Rugerio

Capacitado:

Hernandez Soledad Angel Agustin

Fecha de entrega:
27 de Julio del 2024

Contexto

En el presente documento se pasarán las primeras 40 preguntas de la guia de APX que tengan referencia con Java, se resolverán y se explicaran el por que estas son las respuestas correctas.

Preguntas

1. Which three are bad practices?
    a. Checking for an IOException and ensuring that the program can recover if one occurs.

    b. Checking for ArrayIndexOutOfBoundsException and ensuring that the program can recover if one occurs.

    c. Checking for FileNotFoundException to inform a user that a filename entered is not valid.

    d. Checking for Error and, if necessary, restarting the program to ensure that users are unaware of problems.

    e. Checking for ArrayIndexOutOfBoundsException when iterating through an array to determine when all elements have been visited.

    Explicación:
    La opción **a** es una práctica común y necesaria, ya que las operaciones de entrada/salida pueden fallar por diversas razones (por ejemplo, archivo no encontrado, problema de red, etc.). Atraparlo y recuperarse adecuadamente es una buena práctica.

    Informar al usuario cuando un archivo no se encuentra es una buena práctica, ya que proporciona retroalimentación útil y permite al programa manejar la situación de manera controlada, por lo tanto la opción **c** no es una mala practica.

    La opción **b** es una mala práctica, ya que en lugar de usar excepciones para manejar errores previsibles como un índice fuera de límites, es mejor prevenir estos errores verificando el tamaño del array antes de acceder a él. El manejo de excepciones debe reservarse para situaciones excepcionales y no para el control del flujo normal del programa.

    La opción **d** es tambien una respuesta correcta, ya que los ‘Error’ en Java son problemas graves del entorno de ejecución que generalmente indican que la JVM está en un estado inestable. Intentar atraparlos y recuperarse de ellos puede llevar a un comportamiento impredecible y es una mala práctica. La mejor manera de manejar un ‘Error’ es permitir que el programa falle y arreglar la causa subyacente.

Por último la respuesta **e** muestra una práctica ineficiente y propensa a errores. En lugar de intentar acceder a elementos fuera de los límites y manejar la excepción resultante, es mejor usar un bucle for o foreach para iterar de manera segura sobre los elementos del array. Esto es más claro, más eficiente y evita el uso incorrecto de excepciones para el control del flujo.

2. 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
    a. Mockito
    b. Mock
    c. Inject
    d. InjectMock

    Explicación:
    En el contexto de pruebas unitarias con JUnit y el marco de simulación Mockito, la anotación @Mock se utiliza para indicar que un campo en una clase de prueba es un mock, es decir, una simulación de un objeto real. Esto permite probar el comportamiento de una clase en aislamiento, sin necesidad de crear instancias de sus dependencias reales.

3. Obtenido:

```
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]);
}
```

    ¿Cuál es el resultado en consola?
    a. 3 false 3
    b. 3 false 1
    c. 2 false 1
    d. 3 true 1
    e. 2 true 3

    Explicación:
    Al utilizar el atributo .length en un array de 2 niveles e indicando uno de estos niveles, nos mostrará la cantidad que contiene, por lo tanto array2D[0].length nos dara 3.

    Por otro lado true surge de que .isArray() devuelve un vuelan y evalúa a todo lo de la izquierda, y por lo tanto al ser el array un array, nos devuelve un true.

    Por ultimo en el array [0] [1] encontramos un int 1, y por lo tanto es lo que imprime en consola.A

4. Which two statements are true?

a. An interface CANNOT be extended by another interface.
b. An abstract class can be extended by a concrete class.
c. An abstract class CANNOT be extended by an abstract class.
d. An interface can be extended by an abstract class.
e. An abstract class can implement an interface.
f. An abstract class can be extended by an interface.

Explicación:
a. Una interfaz si puede extender de otra interface
b. Una clase abstracta tambien puede extender de una clase concreta
c. Una clase abstracta puede ser extendida por otra clase abstracta.
d. En Java, una clase (abstracta o concreta) no puede extender (heredar) una interfaz. Solo puede implementar una interfaz.
e. En la programación orientada a objetos, una clase concreta (una clase que puede ser instanciada) puede extender (heredar) una clase abstracta (una clase que no puede ser instanciada directamente y puede contener métodos abstractos que deben ser implementados por las clases derivadas).
f. Una clase abstracta puede implementar una interfaz y no está obligada a proporcionar implementaciones para todos los métodos de la interfaz. Las clases concretas que hereden de esta clase abstracta tendrán que implementar los métodos restantes.

5. Obtenido:

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

¿Cuál es el resultado en consola?

a. Gamma gamma
b. Beta beta
c. Alpha beta
d. Compilation fails

Explicación:
Hay un Downcasting ilegal, ya que new Alpha y new Beta no son instancias de Gamma, y el compilador no permite hacer un cast a una subclase si no hay garantía de que el objeto es una instancia de esa subclase, por lo tanto arroja un error de compilación

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 }
```

a. Private int blipvert(long x) { return 0; }
b. Protected int blipvert(long x) { return 0; }
c. Protected long blipvert(int x, int y) { return 0; }
d. Public int blipvert(int x) { return 0; }
e. Private int blipvert(int x) { return 0; }
f. Protected long blipvert(int x) { return 0; }
g. Protected long blipvert(long x) { return 0; }

Explicación:
Las respuestas correctas permiten compilar el codigo ya que al cambiar el parametro que solicitan son sobrecargas y no sobrescritura, y por tanto pueden funcionar en paralelo sin ningún problema

La respuesta **d** es una sobreescritura pero aumenta la visibilidad sin cambiar nada mas, y por tanto es una sobreescritura valida.

La respuesta **e** es una sobreescritura que disminuye la visibilidad del método, y eso no se puede en java, por tanto esta mal

La respuesta **f** cambia el tipo de retorno al sobreescribirlo, haciendo que no sea valido, ya que Java no lo permite.

7. Dado

```
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. }
```

¿Cuáles dos cambios hechos independientemente, permitirán a Sub compilar?

a. Change line 2 to: public int a;
b. Change line 13 to: public Sub(){ super(5);}

c. Change line 2 to: protected int a;
d. Change line 13 to: public Sub(){ this(5);}
e. Change line 13 to: public Sub(){ super(a);}

Explicación:
El programa no compila ya que en la linea 13, esta intentando hacer referencia a una a que no existe o a la que no tiene acceso, sin embargo, las respuestas **a** y **c** no dan acceso a esta a, ya que son parte del padre, y no son accesibles con this.a ya que deben pertenecer a la propia clase hija, la respuesta **b** utiliza el constructor del padre y le pasa el numero 5, lo cual es algo que la clase Super contempla, y funciona sin problemas, **d** funciona ya que al hacer uso de this, manda a llamar al constructor de la misma clase y esta contempla el paso de una parametro.

Por último **e** no haría al proyecto funcionar, ya que de nuevo, la a no existe en ese scope, y no puede generarla de la nada, provocando error de compilación.

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() { }
}
```

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

Explicación:
Quitando por descarte podríamos decir ninguna de las 4 respuestas restantes son incorrectas, ya que todo lo que dicen que no se debe hacer, o que debe incluir es lo contrario, además la clase no tiene nada que no permita que no se ejecute correctamente.

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(); }
}
```

a. Compilation fails.
b. Atom granite.
c. Granite granite.
d. Atom granite granite.
e. An exception is thrown at runtime.
f. Atom granite atom granite.

Explicación:
Al extender las clases y ejecutar sus constructores, se ejecuta primero el constructor padre, además al usar 'super ' hacemos referencia al constructor padre, por tanto new Rock y super mandan a llamar ambos tanto a el constructor de Atom y el de Rock, y por tanto ejecutarán atom granite, atom granite, dos veces ya que se llaman en dos ocasiones ambos constructores.

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");
        }
}
```

a. one
b. two
c. three
d. four
e. There is no output.

Explicación:
Tanto void main, static void main y void mina, son métodos de la clase MainMethod, por tanto no mandan a llamar a sus println, y el método main, es el unico que lo ejecuta, mostrando en consola “three”.

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();
        }
}
```

a. Cougar c f.
b. Feline cougar c c.
c. Feline cougar c f.
d. Compilation fails.

Explicación:
Al crear un nuevo Cougar se ejecuta el constructor padre mostrando “feline”, después el constructor de la clase actual, mostrando “cougar” y por ultimo ejecuta el método “go()” que imprime type de la clase actual “c” y después el de la clase padre que seria “f”, dando por tanto la respuesta correcta que sería **c**.

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());
        }
}
```

a. Alpha beta
b. Beta beta.

c. Gamma gamma.
d. Compilation fails.

Explicación:
Similar a otra pregunta anterior, hay un Downcasting ilegal, ya que new Alpha y new Beta son padre de Gamma y no sus hijos, y el compilador no permite hacer un cast a una subclase si no hay garantía de que el objeto es una instancia de esa subclase, por lo tanto arroja un error de compilación.

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);
        }
}
```

a. 11
b. 111
c. 1111
d. An exception is thrown at runtime.

Explicación:
Este código ejecuta un error en tiempo de ejecución, ya que la cadena que le pasa al Scanner contiene tanto “a” como un “.” que no son valores int válidos, específicamente en al hacer `accum += s.nextInt()`, intenta leer cuatro enteros de la cadena usando `nextInt()`. Sin embargo, la cadena contiene caracteres no numéricos después del primer entero.

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 ");
        }
}
```

a. The program prints 1 then 2 after 5 seconds.
b. The program prints: 1 thrown to main.
c. The program prints: 1 2 thrown to main.
d. The program prints:1 then t1 waits for its notification.

Explicación:
El programa intentará ejecutar t1.wait(5000); pero lanzará una IllegalMonitorStateException porque el hilo principal no posee el monitor del hilo t1. Esta excepción es capturada en el método main, lo que da como resultado la salida "1 thrown to main".

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);
        }
}
```

a. 420 is the output.
b. An exception is thrown at runtime.
c. All constructors must be declared public.
d. Constructors CANNOT use the private modifier.
e. Constructors CANNOT use the protected modifier

Explicación:
Este código es válido y puede ejecutarse, en el método main, se crea un objeto de la clase ExtendedA, y le pasa como parámetro 420, esté a su vez llama con el método super al constructor de la clase padre “ClassA”, que simplemente asigna este a una variable pública numberOfInstances, este al ser público puede ser llamado con el método print y simplemente lo muestra en consola.

16. The SINGLETON pattern allows:
    a. Have a single instance of a class and this instance cannot be used by other classes
    b. Having a single instance of a class, while allowing all classes have access to that instance.
    c. Having a single instance of a class that can only be accessed by the first method that calls it.

Explicación:
La idea de usar un patrón singleton es que puedas acceder a una sola instancia desde cualquier clase, para no saturar la memoria, por lo que la a y c, no tienen mucho sentido.

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)); }
        }
}
```

a. 111.234 222.567
b. 111.234 222.568
c. 111.234 222.5678
d. An exception is thrown at runtime.

Explicación:
El código intenta convertir cadenas de caracteres a números usando NumberFormat. Sin embargo, el método parse de NumberFormat lanza una ParseException si la cadena de caracteres no se puede analizar correctamente. En el código proporcionado, el arreglo de cadenas sa contiene las siguientes cadenas: "111.234" y "222.5678".

Cuando el NumberFormat intenta analizar la cadena "222.5678", lanzará una ParseException porque está configurado para tener un máximo de tres dígitos fraccionarios (setMaximumFractionDigits(3)). Sin embargo, el análisis real de la cadena no depende de esta configuración; esta configuración solo afecta la forma en que los números se formatean como cadenas, no cómo se analizan las cadenas en números.

La excepción se debe a la presencia de comas en la entrada. En muchos locales, la coma se usa como separador decimal en lugar del punto, lo que puede causar problemas durante el análisis. Para evitar esta excepción, se debe asegurar que las cadenas de entrada estén en el formato correcto para el NumberFormat especificado.

Dado que el código no maneja explícitamente esta excepción, se lanza en tiempo de ejecución, lo que hace que la opción d sea la respuesta correcta.

18. What is the result?

```
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?

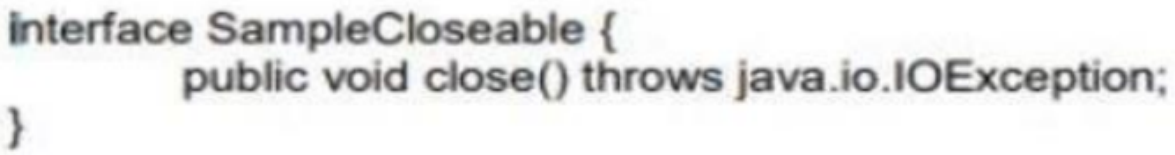
Shape: constructor
Shape: foo
Shape: foo

a. Square square = new Square ("bar"); square.foo ("bar"); square.foo();
b. Square square = new Square ("bar"); square.foo ("bar"); square.foo ("bar");
c. Square square = new Square (); square.foo (); square.foo(bar);
d. Square square = new Square (); square.foo (); square.foo("bar");
e. Square square = new Square (); square.foo (); square.foo ();

Explicación:
Al construir a Square, ya que el constructor del padre ya imprime “shape: constructor”, el crearlo con parámetros, hará que se imprima dos veces esta línea, después llamará a foo sin parámetros, que este a su vez manda a llamar al foo del padre lo que ejecutara Shape: foo, y por ultimo manda a llamar a square.foo con parametros lo que imprime otro Shape: foo.

Por tanto la respuesta correcta es la **d**.

19. Which three implementations are valid?



a. class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }
b. class Test implements SampleCloseable { public void close() throws Exception { // do something } }
c. class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }
d. class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
e. class Test implements SampleCloseable { public void close() { // do something }}

Explicación:
Las respuestas b y d, no son correctas ya que en la b la excepcion declarada en el método close() de Test no es compatible con la declarada en el close() de la interfaz SampleCloseable.

En Java, cuando una clase implementa una interfaz, el método de la clase que implementa un método de la interfaz no puede lanzar una excepción más general que la excepción declarada en la interfaz. En este caso, `Exception` es más general que `java.io.IOException`.

Por otro lado la **d** no es correcta ya que al implementar una interfaz en una clase se usa implements, no extends.

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());
```

a. 2
b. 3
c. 4
d. Compilation fails.

Explicación:
Realmente en el HashMap solamente se agregan 4 elementos con put, por lo que realizar un m.size(), solo va a regresar 4.

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("---");
        }
}
```

a. int x = 4, y = 3, z = 2;
b. int x = 3, y = 2, z = 3;
c. int x = 2, y = 3, z = 3;
d. int x = 2, y = 3, z = 4;
e. int x = 4, y = 2, z= 3;

Explicación:
El ciclo for ejecuta 2 iteraciones (x = 2), dentro de cada iteración del for, el do-while loop ejecuta 3 veces (y = 3), dentro de cada iteración del do-while, el while loop ejecuta 4 veces (z = 4), imprimiendo 1 2 3 4 en cada iteración, después de completar el do-while, se imprime ---. por lo que la respuesta d se adecua.

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();
}
```

a. Line 17
b. Line 18
c. Line 19
d. Line 20
e. Line 21
f. Line 22

Explicación:
Línea 17: problema: Aunque speakIT es una instancia de Tell, está declarado como Speak, que no tiene el método tellItLikeItIs() por sí mismo. Esto dará un error de compilación.

Línea 18: problema: Intentar llamar al método antes de hacer el casting es incorrecto. La llamada al método está fuera del casting. Por lo tanto, esto también dará un error de compilación.

Línea 19: Correcto: Aquí speakIT se castea correctamente a Truth, y como speakIT es en realidad un objeto de Tell (que implementa Truth), la llamada al método es válida.
Línea 20: Correcto: tellIt es un objeto de Tell, y Tell implementa el método tellItLikeItIs(), por lo que esta línea se compila y ejecuta correctamente.
Línea 21: Problema: Similar a la línea 18, el método se está llamando antes de que se realice el casting, lo cual no es correcto en este contexto.

Línea 22: Correcto: tellIt es un objeto de Tell, que implementa Truth. El casting a Truth es correcto y el método se llama exitosamente.

23. What is the result?

```
class Feline {
    public String type = "f";
    public Feline() {
        System.out.print(s: "feline ");
    }
}
public class Cougar extends Feline{
    public Cougar() {
        System.out.print(s: "cougar ");
    }
    void go(){
        String type = "c";
        System.out.print(this.type + super.type);
    }

    Run | Debug
    public static void main(String[] args) {
        new Cougar().go();
    }
}
```

a. Feline cougar c f
b. Feline cougar c c
c. Feline cougar f f
d. No compila

Explicación:
La respuesta es Feline cougar, porque feline es llamado por el constructor padre, cougar por el constructor hijo, y this.type manda a llamar a type de Cougar que lo hereda del padre (diferente a si solo colocara type), y super.type del padre (que es curiosamente el mismo), por lo que daría f f

24. ¿Qué acción realiza el archivo de dependencia pom.xml?*

a. Revisa que versiones de dependencias se tienen con otros proyectos
b. Elimina las dependencias con otros proyectos
c. Recupera todas las dependencias con otros proyectos

d. Modifica las dependencias que se tienen con otros proyectos

Explicación:
El archivo pom.xml en Maven se utiliza para manejar las dependencias del proyecto, configurar plugins, y definir otros aspectos de la construcción del proyecto. La respuesta correcta es a. y c. ya que pom.xml revisa y recupera todas las dependencias necesarias para el proyecto.

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

    a. Datasource
    b. JPA
    c. JDBC
    d. Database

    Explicación:
    Las instancias de Datasource son generalmente gestionadas por el servidor de aplicaciones o el contenedor de Spring, y se inyectan en la aplicación a través de configuraciones o inyecciones de dependencias.

    Con Datasource, las conexiones son administradas automáticamente. El pool de conexiones maneja la apertura y cierre de conexiones, permitiendo que la aplicación solicite y libere conexiones sin preocuparse por su ciclo de vida.

26. En un Batch, ¿Cuál es el step donde el desarrollador introducirá la lógica empresarial necesaria?*

    a. itemWriter
    b. Task
    c. Chunk
    d. JobLauncher

    Explicación:
    En un proceso de batch, la lógica empresarial necesaria se introduce en un Task. El Task es una unidad de trabajo específica dentro de un step del job batch. Mientras que otros componentes como el JobLauncher solo lanzan el trabajo y el ItemWriter se encarga de escribir los datos procesados, el Task es el lugar donde el desarrollador puede implementar la lógica empresarial necesaria para realizar las operaciones requeridas en ese step. Si el step es de tipo chunk, la lógica empresarial puede ir en el ItemProcessor, pero en el caso de step tipo tasklet, el Task es el componente clave para dicha lógica.

27. El identificador único para la ejecución de transacciones/Job se genera automáticamente como último paso cuando se invoca las transacciones en línea/Batch.
    El resultado de la transacción se puede propagar para la correlación en las otras

capas.

a. Falso
b. Verdadero

Explicación:
En la ejecución de transacciones o jobs en un entorno batch, un identificador único (a menudo llamado JobExecutionId o similar) se genera automáticamente como el último paso cuando se invoca el trabajo. Este identificador es esencial para rastrear y gestionar la ejecución del job, permitiendo una correlación efectiva con otros procesos y capas del sistema.

El resultado de la transacción, que incluye este identificador único, se puede propagar a otras capas del sistema para mantener la coherencia y permitir la correlación entre las diferentes partes del sistema. Esto es crucial para el monitoreo, auditoría y diagnóstico de problemas, ya que permite vincular la ejecución del job batch con otras actividades y servicios que interactúan con él.

28. 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.

    a. Reutilizando código
    b. Anti-patrón
    c. Metodología
    d. Patrón de diseño

    Explicación:
    Un patrón de diseño es una solución general, reutilizable, y comprobada a un problema común en un contexto de diseño de software. Los patrones de diseño son documentados y compartidos porque han demostrado ser efectivos en la resolución de problemas específicos de diseño y son aplicables en múltiples contextos. Los patrones de diseño facilitan la creación de software más modular, mantenible y comprensible al proporcionar un lenguaje común y soluciones predefinidas para problemas recurrentes.

29. ¿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);
        }
}
```

a. [7, 5]
b. [7, 1]
c. [7, 5, 1]
d. [7, 1, 5, 1]

Explicación:
Al utilizar el método remove, eliminas la posición 1, no el valor 1, por tanto cuando tenías 7151 elimina el primer 1, y por eso nos da como resultado 7, 5, 1.

30. 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 }
```

a. Public int blipvert(int x) { return 0; }
b. Protected long blipvert(int x) { return 0; }
c. Protected int blipvert(long x) { return 0; }
d. Private int blipvert(long x) { return 0; }
e. Protected long blipvert(int x, int y) { return 0; }
f. Private int blipvert(int x) { return 0; }
g. Protected long blipvert(long x) { return 0; }

Explicación:
La f no compila por que reduce la visibilidad de el método de Blip, y esto no se puede al extender de una clase, y la b no es valida por que no se puede cambiar el tipo de retorno al sobreescribir un método, por lo que por descarte, son todas las demás.

31. La información utilizada en la lógica de negocio de la transacción debe incluirse en _____.

    a. Los archivos locales.
    b. Las variables globales.
    c. Los parámetros de entrada.
    d. Las variables locales.

    Explicación:
    Los parámetros de entrada son los valores que se pasan a una función o método cuando se llama a este. En el contexto de la lógica de negocio de una transacción, los parámetros de entrada permiten que la función o método reciba los datos necesarios para realizar su operación de manera segura y eficaz. Al utilizar parámetros de entrada, se mejora la modularidad y la reutilización del código, y se evita el uso excesivo de variables globales o locales que podrían complicar el seguimiento y la gestión de los datos. Además, los parámetros de entrada permiten que la lógica de negocio sea más clara y explícita en cuanto a los datos que necesita para ejecutarse correctamente.

32. Estos tipos de librerías se pueden combinar entre sí para tener una librería con diferentes capacidades o no elegir ninguno de las opciones anteriores.

    a. Online y Batch
    b. Shell y JDBC

    Explicación:
    Las librerías online y batch se pueden combinar para crear aplicaciones con diferentes capacidades. Una librería online maneja transacciones en tiempo real, permitiendo la interacción directa con el usuario y la respuesta inmediata a las solicitudes. Por otro lado, una librería batch procesa grandes cantidades de datos en segundo plano sin necesidad de interacción del usuario, generalmente programada para ejecutarse en momentos específicos.

    Al combinar estas dos librerías, una aplicación puede aprovechar las ventajas de ambas: la capacidad de responder a las solicitudes del usuario en tiempo real y la capacidad de realizar procesamiento masivo y tareas automatizadas en el fondo. Esto permite una arquitectura más flexible y robusta, capaz de manejar diferentes tipos de cargas de trabajo y necesidades de procesamiento.

    Shell y JDBC no son tipos de librerías que se combinen de la misma manera. Shell se refiere a una interfaz de línea de comandos, mientras que JDBC es una API para conectar y ejecutar consultas en bases de datos en Java. Aunque pueden

interactuar en un entorno de desarrollo, no se combinan para formar una única librería con capacidades mixtas.

33. Es una contradicción al usar el patrón CRUD

    a. Mantener todas las operaciones sobre la entidad encapsuladas en la librería
    b. Tener varias librerías parciales para una única entidad
    c. Centralizar en una librería de todas las operaciones básicas sobre una entidad
    d. Tener una implementación ligera, evitando extender la lógica más allá del objetivo de la operación

    Explicación:
    El patrón CRUD (Crear, Leer, Actualizar, Borrar) se basa en tener todas las operaciones básicas para manipular una entidad centralizadas y encapsuladas en una única librería o módulo. Esto facilita la gestión y el mantenimiento del código, asegurando que todas las operaciones relacionadas con una entidad específica estén en un solo lugar.

    Tener varias librerías parciales para una única entidad contradice este principio, ya que dispersa las operaciones CRUD en múltiples lugares. Esto puede llevar a una mayor complejidad, dificultar el mantenimiento del código, y aumentar la posibilidad de inconsistencias o errores, ya que las operaciones sobre una misma entidad no están centralizadas ni bien organizadas. Mantener todas las operaciones sobre la entidad encapsuladas en una sola librería promueve una implementación más ligera y centralizada, alineada con el objetivo del patrón CRUD.

34. ¿Qué es Batch?
    a. No conozco la respuesta
    b. Proceso que varía con respecto a sus dependencias
    c. Proceso idéntico a online
    d. Proceso que reemplaza al mainframe
    e. Es un patrón de ejecución, ejecución de un programa sin el control o supervisión directa del usuario

    Explicación:
    Un proceso batch (o procesamiento por lotes) se refiere a la ejecución de programas o tareas sin la necesidad de control o supervisión directa del usuario. Estos procesos se ejecutan en segundo plano, típicamente programados para ejecutarse en momentos específicos (como durante la noche o en períodos de baja actividad) y están diseñados para manejar grandes volúmenes de datos o realizar tareas repetitivas.

    El procesamiento batch es diferente al procesamiento en línea, que requiere la interacción directa y continua del usuario. Los procesos batch son esenciales para tareas como la generación de informes, la actualización de bases de datos, el

procesamiento de transacciones en masa, y otros trabajos que pueden ser agrupados y ejecutados secuencialmente sin intervención humana directa.

Por lo tanto, un batch no es un proceso que varía con respecto a sus dependencias, ni es idéntico a online, ni reemplaza al mainframe. Es un patrón de ejecución diseñado para operar sin supervisión directa.

35. En un DTO se desea intercambiar información de forma

    a. No conozco la respuesta
    b. Coherente
    c. Agrupada
    d. Asociada
    e. Organizada

    Explicación:
    Un DTO (Data Transfer Object) se utiliza para intercambiar información de forma organizada. Los DTOs son objetos simples que contienen solo datos y carecen de lógica de negocio. Su propósito es encapsular los datos que se desean transferir entre diferentes capas de una aplicación, como entre la capa de presentación y la capa de negocio, o entre la capa de negocio y la capa de acceso a datos.

    Al usar DTOs, los datos se estructuran y organizan de manera clara y coherente, facilitando el intercambio de información entre distintos componentes del sistema. Esto permite una mejor separación de preocupaciones y una mayor mantenibilidad del código, ya que los datos se agrupan de manera lógica y se transfieren en un formato bien definido.

36. Ejemplos de patrones de diseño

    a. No conozco la respuesta
    b. DTO en componentes APX.
    c. Todas
    d. Paginación en bibliotecas.
    e. Paginación en transacciones.
    f. CRUD en bibliotecas.

    Explicación:
    Los patrones de diseño abarcan una variedad de soluciones reutilizables a problemas comunes en el desarrollo de software. Aquí tienes ejemplos de cómo algunos de ellos se aplican:

    DTO en componentes APX: DTO (Data Transfer Object) es un patrón de diseño que se usa para transferir datos entre diferentes capas de una aplicación, especialmente cuando se utilizan frameworks como APX (por ejemplo, en BBVA). Su propósito es encapsular y organizar datos en objetos que se transfieren entre la capa de presentación y la capa de negocio.

Paginación en bibliotecas: La paginación es un patrón que se utiliza para dividir grandes conjuntos de datos en páginas más pequeñas, lo que mejora la eficiencia y la usabilidad al presentar datos en interfaces de usuario o al procesar grandes volúmenes de información en bibliotecas o sistemas de bases de datos.

Paginación en transacciones: Similar a la paginación en bibliotecas, este patrón se utiliza para gestionar y procesar datos en bloques más pequeños durante transacciones o procesos que requieren la manipulación de grandes volúmenes de datos.

CRUD en bibliotecas: CRUD (Crear, Leer, Actualizar, Borrar) es un patrón de diseño que define las operaciones básicas que se pueden realizar sobre los datos en una aplicación. Es común en el diseño de bibliotecas de acceso a datos y en la implementación de servicios de datos.

Cada uno de estos ejemplos representa una aplicación práctica de un patrón de diseño que ayuda a resolver problemas específicos en el desarrollo de software. Por lo tanto, la respuesta correcta es Todas.

37. Batch consta de 3 partes ¿Cuáles son?

    a. Salida: escribe o realiza alguna acción como resultado del procesamiento.
    b. No conozco la respuesta
    c. Procesamiento: se realiza algún tipo de lógica empresarial sobre estos datos de entrada.
    d. Entrada: obtenga y lea la información de entrada al proceso.
    e. Otro:

    Explicación:
    Entrada: Esta fase implica la obtención y lectura de los datos necesarios para el proceso batch. Los datos de entrada pueden provenir de diversas fuentes, como archivos, bases de datos o sistemas externos.

    Procesamiento: En esta etapa, se aplica la lógica empresarial a los datos de entrada. Aquí es donde se realizan las operaciones necesarias, como transformaciones, cálculos, o cualquier otra manipulación de los datos, según los requisitos del proceso batch.

    Salida: Una vez que los datos han sido procesados, esta fase se encarga de escribir o realizar alguna acción con los resultados del procesamiento. Esto puede incluir la generación de informes, la actualización de bases de datos, o el envío de datos a otros sistemas.

38. ¿En qué ayuda un DTO?

    a. Ayuda a agrupar métodos de un componente
    b. Ayuda a identificar los métodos que se repiten en componentes APX

c. El uso de este patrón ayuda a identificar las agrupaciones de datos simples y/o complejos, y evita duplicidad de código.
d. No conozco la respuesta

Explicación:
Un DTO (Data Transfer Object) es un patrón de diseño que se utiliza para transferir datos entre diferentes partes de una aplicación, como entre la capa de presentación y la capa de negocio o la capa de negocio y la capa de persistencia.

El principal objetivo de un DTO es agrupar datos simples y/o complejos en un único objeto que encapsula la información que se desea transferir. Esto ayuda a estructurar los datos de manera organizada y coherente. Al utilizar DTOs, se evita la duplicidad de código y se simplifica el proceso de transferencia de datos al mantener una estructura clara y definida.

39. ¿Qué es patrón de diseño?

a. Patrón de identificación de procesos
b. Proporciona una solución a un problema de diseño
c. Identificador de diseños
d. Proporciona información de la ejecución
e. No conozco la respuesta
f. Patrón de identificación único de procesos

Explicación:
Un patrón de diseño es una solución general y reutilizable a un problema común en el diseño de software. Estos patrones son soluciones probadas y documentadas que ayudan a resolver problemas de diseño específicos y son aplicables a diferentes situaciones y contextos en el desarrollo de software.

Los patrones de diseño no son implementaciones concretas, sino más bien descripciones y plantillas para resolver problemas que se presentan frecuentemente en el desarrollo de software. Estos patrones permiten a los desarrolladores aplicar soluciones efectivas y comprobadas a problemas de diseño comunes, mejorando la calidad y la mantenibilidad del código.

40. ¿Para qué se usa el JPA?

a. Permite insertar, actualiza y consultar en BBDD
b. No conozco la respuesta
c. Permite realizar un almacenamiento en caché de tablas de BBDD
d. Permite manejar las utilidades de APX (Datio, JDBC,...)
e. Permite usar componentes APX con cualquier BBDD

Explicación:

JPA (Java Persistence API) es una especificación en Java que se utiliza para gestionar la persistencia de datos en aplicaciones Java. Permite a los desarrolladores realizar operaciones de inserción, actualización, eliminación y consulta en bases de datos relacionales de manera eficiente y sencilla, utilizando un modelo de objetos en lugar de interactuar directamente con la base de datos mediante SQL.

JPA proporciona una capa de abstracción sobre el acceso a datos y facilita la integración con bases de datos mediante la representación de datos como entidades Java. Esto permite que las aplicaciones manejen datos en términos de objetos y relaciones entre estos objetos, simplificando la gestión de la persistencia y mejorando la mantenibilidad del código.

41. ¿Qué es un DTO?*
    a. Agrupa métodos de un componente
    b. Un objeto de transferencia de datos es, transporta datos entre procesos
    c. Es un objeto de agrupación de componentes APX
    d. No conozco la respuesta

Explicación:
Un DTO (Data Transfer Object) es un patrón de diseño que se utiliza para transportar datos entre diferentes partes de una aplicación o entre aplicaciones. Un DTO es un objeto simple que contiene solo datos y carece de lógica de negocio. Su propósito principal es agrupar y encapsular la información que se desea transferir, facilitando el intercambio de datos entre capas o sistemas.

Los DTOs se utilizan para mejorar la eficiencia y la claridad al transferir datos, evitando la necesidad de exponer directamente los objetos del dominio o entidades del modelo de datos. Al agrupar los datos relevantes en un único objeto, los DTOs ayudan a reducir la complejidad y mejorar el rendimiento, especialmente en aplicaciones distribuidas y sistemas de comunicación entre procesos.