

Preguntas

56. Son patrones de diseño de microservicios

- a) Circuit Breaker, Adaptative Lifo, MQ Strategy
- b) System, Process y Client
- c) Retry, Circuit Breaker, Adaptative Lifo y Bulkhead.**
- d) Ninguna de las anteriores

57. ¿Qué afirmaciones son verdaderas tanto para las clases abstractas como para las interfaces? (Elije todas las correctas)

- a) Ambos pueden contener métodos estáticos.**
- b) Ambos se pueden ampliar con la clave extend.**
- c) Ambos pueden contener métodos predeterminados.
- d) Ambos heredan de java.lang.Object.
- e) Ninguno de los dos puede ser instanciado directamente.**
- f) Ambos pueden contener variables finales estáticas públicas.**
- g) Supone que todos los métodos dentro de ellos son abstractos.

58. ¿Cuál no es un objetivo de Maven?

- a) Clean
- b) Package
- c) Debug**
- d) Install

59. ¿Si deseas obtener una copia de un repositorio Git existente en un servidor qué comando se utiliza?

- a) git commit
- b) git log
- c) git clone**
- d) git add

60. ¿Qué es un repositorio remoto en Git?

Un repositorio remoto en Git es una versión de tu proyecto que se encuentra alojada en un servidor o servicio externo, fuera de tu entorno local de desarrollo. Este repositorio permite que múltiples desarrolladores colaboren en el mismo proyecto, compartiendo y sincronizando sus cambios. Los repositorios remotos son esenciales para el trabajo en equipo y la gestión de proyectos distribuidos.

61. Dada la siguiente clase

```
public class Helper {  
    public static < U extends Exception > void  
        printException(U u){  
        System.out.println(u.getMessage());  
    }  
  
    public static void main(String[] args) {  
        //línea 9  
    }  
}
```

¿Cuál de las siguientes instrucciones puede colocarse en la línea 9 para que la clase Helper compile?

- a) **Helper.printException(new Exception("B"));**
- b) **Helper. printException(new FileNotFoundException("A"));**
- c) Helper.<Throwable>printException(new Exception("C"));
- d) **Helper.<NullPointerException>printException(new NullPointerException("D"));**
- e) Helper. printException(new Throwable("E"));

62. ¿Cuál es la salida al ejecutar el siguiente código?

```
public class Fish {  
    public static void main(String[] args) {  
        int numFish = 4;  
        String fishType = "tuna";  
        String anotherFish = numFish + 1;  
        System.out.println(anotherFish + " " + fishType);  
        System.out.println(numFish + " " + 1);  
    }  
}
```

- a) 51tuna
- b) 5tuna
- c) 5
- d) 41
- e) 5 tuna
- f) 4 1
- g) **El código no compila**

63. ¿Cuál de las siguientes opciones son correctas? (Elija todas las correctas)

```
public class StringBuilders {  
    // usage  
    public static StringBuilder work(StringBuilder a, StringBuilder b){  
        a = new StringBuilder("a");  
        b.append("b");  
        return a;  
    }  
  
    public static void main(String[] args) {  
        StringBuilder s1 = new StringBuilder("s1");  
        StringBuilder s2 = new StringBuilder("s2");  
        StringBuilder s3 = work(s1,s2);  
        System.out.println("s1 = " + s1);  
        System.out.println("s2 = " + s2);  
        System.out.println("s3 = " + s3);  
    }  
}
```

- a) s2 = s2
- b) s3 = null
- c) s1 = s1**
- d) s3 = a**
- e) El código no compila
- f) s2 = s2b**
- g) s1 = a

64. ¿A qué hace referencia el principio de Liskov?

Este principio establece que en un sistema de herencia, los objetos de una clase derivada (subclase) deben ser sustituibles por objetos de la clase base (superclase) sin alterar las propiedades del sistema (correctitud, funcionamiento).

65. ¿Qué es un “code smell”?

Un code smell es un indicio de que algo puede estar mal en el código. No son errores que impidan que el código funcione, sino características que sugieren que el diseño del código puede ser mejorado para evitar futuros problemas. Los code smells suelen señalar debilidades en el diseño que podrían ralentizar el desarrollo o aumentar el riesgo de errores y fallos en el futuro.

66. ¿Qué significa el acrónimo CRUD en una API REST?

Create, Read, Update, Delete

67. ¿Para qué nos sirve utilizar un profile dentro del archivo pom.xml?

Un perfil dentro del archivo pom.xml es una manera de personalizar la construcción del proyecto para diferentes entornos o escenarios. Los perfiles permiten modificar los parámetros de construcción, dependencias, plugins y otras configuraciones de Maven según sea necesario para distintas situaciones.

68. Dadas las siguientes clases Vehicle y Car

```
package my.vehicles;

public class Vehicle {
    public String make;
    protected String model;
    private int year;
    int mileage;
}

package my.vehicles.cars;

import my.vehicles.*;

public class Car extends Vehicle {
    public Car() {
        //línea 7
    }
}
```

¿Cuál de las siguientes instrucciones pueden colocarse en la línea 7 para que la clase Car compile correctamente? (Seleccione las que apliquen)

- a) mileage = 15285;
- b) Ninguna de las anteriores.
- c) make = "Honda";
- d) year = 2009;
- e) model = "Pilot";

69. Enumere cuatro interfaces de la API colecciones

- a) List, Map, Set, Queue.
- b) ArrayList, Map, Set, Queue.
- c) List, HashMap, HashSet, PriorityQueue.
- d) List, Map, HashSet, PriorityQueue.

70. Selecciona la respuesta correcta con respecto al resultado del bloque de código.

```
public class Test5 {  
    public static void main(String args[]) {  
        Side primerIntento = new Head();  
        Tail segundoIntento = new Tail();  
        Coin.overload(primerIntento);  
        Coin.overload((Object)segundoIntento);  
        Coin.overload(segundoIntento);  
        Coin.overload((Side)primerIntento);  
    }  
}  
  
interface Side { String getSide();}  
  
class Head implements Side {  
    public String getSide() { return "Head ";}  
}  
  
class Tail implements Side {  
    public String getSide() { return "Tail ";}  
}  
  
class Coin {  
    public static void overload(Head side) {System.out.println(side.getSide());}  
    public static void overload(Tail side) {System.out.println(side.getSide());}  
    public static void overload(Side side) {System.out.println("Side ");}  
    public static void overload(Object side) {System.out.println("Object ");}  
}
```

- a) Head
Object
Tail
Side
- b) No compila
- c) Side
Object
Tail
Side
- d) Head
Head
Tail
Tail
- e) Side
Head
Tail
Side

71. ¿Cuál es la salida al ejecutar el siguiente código?

```
public class Lion {  
    public void roar(String roar1, StringBuilder roar2) {  
        roar1.concat("!!!");  
        roar2.append("!!!");  
    }  
    public static void main(String[] args) {  
        String roar1 = "roar";  
        StringBuilder roar2 = new StringBuilder("roar");  
        new Lion().roar(roar1, roar2);  
        System.out.println(roar1 + " " + roar2);  
    }  
}
```

- a) **roar roar!!!**
- b) roar!!! Roar
- c) Se lanza una excepción
- d) roar!!! roar!!!
- e) roar roar
- f) El código no compila

72. ¿Cuál de los siguientes es cierto acerca de una subclase concreta?

- a) Una subclase concreta no se puede marcar como final.
- b) Una subclase concreta debe implementar todos los métodos definidos en una interfaz heredada.
- c) **Una subclase concreta debe implementar todos los métodos abstractos heredados.**
- d) Una subclase concreta puede declararse como abstracta.
- e) Los métodos abstractos no pueden ser anulados por una subclase concreta.

73. ¿Cuál es la salida del siguiente código?

```
1 public abstract class Catchable {
2     protected abstract void catchAnObject(Object x);
3
4     public static void main(String [] args) {
5         java.util.Date now = new java.util.Date();
6         Catchable target = new MyStringCatcher();
7         target.catchAnObject(now);
8     }
9 }
10
11 class MyStringCatcher extends Catchable {
12     public void catchAnObject(Object x) {
13         System.out.println("Caught object");
14     }
15
16     public void catchAnObject(String s) {
17         System.out.println("Caught string");
18     }
19 }
```

- a) Error de compilación línea 12
- b) Error compilación línea 16
- c) Caught string
- d) Error compilación línea 2
- e) **Caught Object**

74. Seleccione la respuesta que considere correcta, dado el siguiente bloque de código.

```
1 import java.util.Arrays;
2 import java.util.List;
3
4 public class Example {
5
6     public static void main(String[] args) {
7         List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
8
9         int result = numbers.stream()
10             .filter(n -> n % 2 == 0)
11             .reduce(0, (a, b) -> a + b);
12
13         System.out.println(result);
14     }
15 }
16
17 }
```

- a) 3
- b) 9
- c) 14
- d) **6**

75. ¿Qué declaración representa una declaración válida que permitirá la inclusión de clases del paquete java.util?

- a) #include java.util.*;
- b) #include java.util;
- c) import java.util.*;
- d) **import java.util;**

76. ¿Qué es la cobertura de código?

La cobertura de código es una métrica utilizada en el desarrollo de software para medir el grado en que el código fuente de un programa ha sido ejecutado durante las pruebas. Esta métrica ayuda a los desarrolladores a identificar qué partes del código no están siendo probadas y, por lo tanto, pueden contener errores no detectados.

77. ¿Cuál es el formato correcto para hacer un commit en Git?

- a) Descripción breve del cambio y nombre del autor.
- b) **Tipo de cambio, descripción breve, cuerpo opcional y notas de pie de página.**
- c) Solo se necesita una breve descripción del cambio.
- d) Nombre de la rama, descripción detallada del cambio y fecha.

78. ¿Qué es el patrón de diseño Singleton y cómo se implementa en Java 8?

Su propósito es restringir la instanciación de una clase a un solo objeto. Este patrón es útil cuando exactamente una instancia de clase es necesaria para coordinar acciones en un sistema.

79. En los verbos REST ¿Cuál es la diferencia en el uso de PATCH y PUT?

PUT requiere se le envíe la entidad completa mientras que PATCH solo los atributos a modificar.

80. ¿Cuál es la diferencia entre las anotaciones: @RestController, @Component, @Service y @Repository?

No existe diferencia funcional entre ellas sino semántica, las 4 son anotaciones de Spring que crean un bean y lo agregan al contexto de Spring.

81. ¿Cuál es una buena práctica al escribir pruebas unitarias?

- a) Ejecutar pruebas con poca frecuencia.
- b) Asegurarse de que las pruebas sean claras y concisas.**
- c) Probar solo una pequeña parte de una función.
- d) Hacer que las pruebas dependan de otras pruebas.

82. ¿Cuál es la ventaja de usar APIs REST sobre otros tipos de servicios web?

Las APIs REST tienen varias ventajas sobre otros tipos de servicios web:

Simplicidad: Usan HTTP y formatos simples como JSON, facilitando su uso y comprensión.

Flexibilidad y Escalabilidad: Permiten un desarrollo desacoplado entre cliente y servidor, facilitando la escalabilidad horizontal.

Interoperabilidad: Funcionan en múltiples plataformas y lenguajes, gracias al uso de estándares abiertos.

Eficiencia: Permiten el uso de cachés HTTP y minimizan el uso de ancho de banda.

Mantenimiento: Son más fáciles de mantener y versionar, con herramientas como Swagger para la documentación automática.

83. Selecciona la respuesta correcta con respecto al resultado del bloque de código.

```
public class Test4 {  
    public static void main(String[] args) {  
        List list = Arrays.asList(25,7,25,67);  
        System.out.println(list);  
        System.out.println(new HashSet(list));  
        System.out.println(new TreeSet(list));  
        System.out.println(new HashSet(list));  
        System.out.println(new ConcurrentSkipListSet(list));  
    }  
}
```

- a) No compila
- b) [25, 7, 25, 67]**
[67, 7, 25]
[7, 25, 67]
[67, 7, 25]
[7, 25, 67]
- c) [25, 7, 67]
[67, 7, 25]
[7, 25, 67]
[67, 7, 25]
[7, 25, 67]
- d) [67, 7, 25]
[67, 7, 25]
[67, 7, 25]

- [67, 7, 25]
[67, 7, 25]
e) [25, 7, 25, 67]
[7, 25, 67]
[67, 7, 25]
[7, 25, 67]
[67, 7, 25]

84. ¿Cuáles son los 4 pilares de la programación orientada a objetos?

Polimorfismo, Abstracción, Herencia y Encapsulamiento.

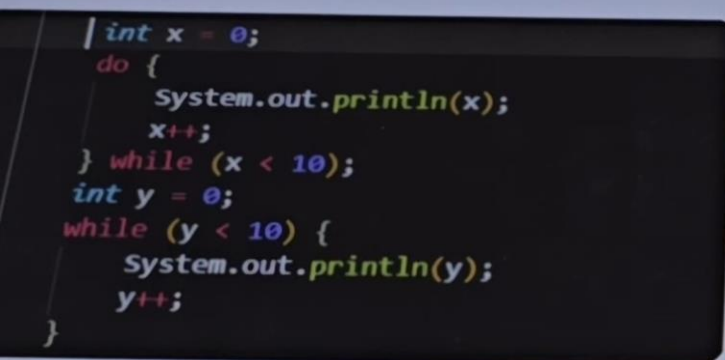
85. ¿Qué utilidad de línea de comandos basada en MS Windows le permitirá ejecutar el intérprete de Java sin abrir la ventana de la consola?

La utilidad de línea de comandos en MS Windows que permite ejecutar el intérprete de Java sin abrir la ventana de la consola se llama javaw.exe. javaw.exe es similar a java.exe, que se utiliza para ejecutar aplicaciones Java desde la línea de comandos, pero javaw.exe se ejecuta en segundo plano y no abre una ventana de consola.

86. ¿Qué es un endpoint en una API REST?

Un endpoint en una API REST es un punto de acceso específico que define una operación sobre un recurso en particular. Es la URL (Uniform Resource Locator) a la que se puede enviar una solicitud para interactuar con un servicio web o una aplicación web que sigue el estilo arquitectónico REST (Transferencia de Estado Representacional).

87. ¿Cuál es el valor de x e y al final el programa?



```
int x = 0;
do {
    System.out.println(x);
    x++;
} while (x < 10);
int y = 0;
while (y < 10) {
    System.out.println(y);
    y++;
}
```

- a) X=9 y=10
b) X=10 y=9
c) X=10 y=10
d) X=9 y=9

88. ¿Dado el siguiente enum y clase cuál es la opción que puede ir en el espacio en blanco para que el código compile?

```
enum Season { SPRING, SUMMER, WINTER }
public class Weather {
    public int getAverageTemperate(Season s) {
        switch (s) {
            default:
                _____ return 30;
        }
    }
}
```

a) Ninguno de los anteriores

- b) case SUMMER ->
- c) case Season.Winter:
- d) case FALL:
- e) case Winter, Spring:
- f) case SUMMER | WINTER:

89. ¿Cuál es el resultado de compilar y ejecutar el siguiente programa?

```
public static void main(String[] args) {
    boolean stmt1 = "champ" == "champ";
    boolean stmt2 = new String( original: "champ") == "champ";
    boolean stmt3 = new String( original: "champ") == new String( original: "champ");
    System.out.println(stmt1 && stmt2 || stmt3);
}
```

a) False

- b) no se produce salida
- c) true
- d) error de compilación

90. ¿Cómo se manejan las excepciones en Java?

Las excepciones se manejan con bloques try catch finally en Java. La excepción trae with Resources es una forma de cerrar automáticamente los recursos abiertos en un bloque try.

91. ¿Qué clase del paquete java.io permite leer y escribir archivos en ubicaciones específicas dentro de un archivo?

La clase del paquete java.io que permite leer y escribir archivos en ubicaciones específicas dentro de un archivo se llama RandomAccessFile. Esta clase proporciona métodos para leer y escribir bytes desde y hacia cualquier ubicación dentro de un archivo.

92. Todas las siguientes definiciones de clases my School classroom y my City School ¿qué números de línea en el método main generan un error de compilación? (Elija todas las opciones correctas)

```
1: package my.school;
2: public class Classroom {
3:     private int roomNumber;
4:     protected String teacherName;
5:     static int globalKey = 54321;
6:     public int floor = 3;
7:     Classroom(int r, String t) {
8:         roomNumber = r;
9:         teacherName = t; } }

1: package my.city;
2: import my.school.*;
3: public class School {
4:     public static void main(String[] args) {
5:         System.out.println(Classroom.globalKey);
6:         Classroom room = new Classroom(101, "Mrs. Anderson");
7:         System.out.println(room.roomNumber);
8:         System.out.println(room.floor);
9:         System.out.println(room.teacherName); } }
```

- a) Ninguna, el código compila bien
b) línea 6
c) línea 9
d) línea 7
e) línea 8
f) línea 5

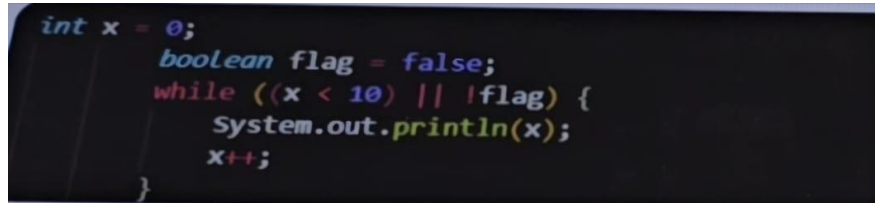
```
package my.city;
import my.school.*;
public class School {
    public static void main(String[] args) {
        System.out.println(Classroom.globalKey);
        Classroom room = new Classroom(101, "Mrs. Anderson");
        System.out.println(room.roomNumber);
        System.out.println(room.floor);
        System.out.println(room.teacherName);
    }
}
```

Aquí dejo el código que muestra las líneas que tienen error.

93. ¿Qué es una expresión lambda en Java?

Una expresión lambda en Java es una forma concisa de representar una función anónima, es decir, una función sin nombre que puede ser creada "en el momento" y pasada como argumento a otro método o almacenada en una variable. Las expresiones lambda fueron introducidas en Java 8 como parte de las características del lenguaje para facilitar la programación funcional

94. ¿Qué hace el siguiente código fuente?



```
int x = 0;
boolean flag = false;
while ((x < 10) || !flag) {
    System.out.println(x);
    x++;
}
```

- a) Muestra los números del 1 al 10
- b) muestra un 10
- c) se queda en un bucle infinito**
- d) muestra los números del 0 al 9

95. ¿Qué son las anotaciones en java?

Las anotaciones en Java son metadatos que proporcionan información adicional sobre elementos del código, como clases, métodos, variables y otros elementos. Se utilizan para agregar información descriptiva, configuración o comportamiento especial a los elementos del programa. Las anotaciones se definen mediante la palabra clave @, seguida del nombre de la anotación.

96. ¿Qué son las expresiones regulares en Java?

Las expresiones regulares en Java son secuencias de caracteres que forman un patrón de búsqueda. Estas secuencias se utilizan para buscar y manipular texto basándose en patrones específicos. Las expresiones regulares son extremadamente poderosas y flexibles, lo que las convierte en una herramienta fundamental para el procesamiento de cadenas de texto en Java y en muchos otros lenguajes de programación.

97. ¿Qué es una anotación en Spring?

las anotaciones son etiquetas especiales que se utilizan para proporcionar metadatos y configuración adicional a los componentes y clases dentro de una aplicación Spring. Estas anotaciones permiten la configuración de manera declarativa, lo que significa que puedes expresar la configuración directamente en el código fuente de la aplicación en lugar de hacerlo a través de archivos de configuración XML o de otro tipo.

98. ¿Cuál es la salida?

```
import java.util.ArrayList;

public class OtherExample {
    public static void main(String[] args) {
        var list = new ArrayList<String>();
        list.add("Austin");
        list.add("Boston");
        list.add("San Francisco");
        var c :long = list.stream()
            .filter(a -> a.length() > 10) //línea x
            .count();
        System.out.println(c + " " + list.size());
    }
}
```

- a) Ninguna de las anteriores
- b) 1 3**
- c) 1 1
- d) El código no compila en la línea x
- e) 2 3

99. ¿Cuál es la salida de la siguiente aplicación?

```
interface Speak { default int talk(){ return 7;} }
interface Sing { default int talk(){ return 5; }}

public class Performance implements Speak, Sing {
    public int talk(String... x) {
        return x.length;
    }

    public static void main(String[] notes) {
        System.out.println(new Performance().talk());
    }
}
```

- a) 5
- b) 7
- c) El código compila sin problemas la salida no se puede determinar hasta el tiempo de ejecución.
- d) Ninguna de las anteriores.
- e) **El código no compila**

100. ¿Qué conjunto de modificadores, cuando son agregados a un método default dentro de una interfaz, evitan que sea sobrescrito por la clase que lo implementa?

- a) private
- b) const
- c) **final**
- d) private static
- e) Ninguna de las anteriores
- f) Static

101. ¿Qué tipo de excepción se produce cuando se intenta realizar una operación incompatible con el tipo de datos en Java?

En Java, cuando se intenta realizar una operación incompatible con el tipo de datos, se produce una `java.lang.ClassCastException`. Esta excepción se lanza para indicar que el código ha intentado convertir un objeto a una subclase de la cual no es una instancia.

102. ¿Qué es un starter?

a) Es una herramienta que nos facilita la creación y configurar un proyecto cargando las dependencias necesarias para un objetivo.

b) Es el inicializador de un proyecto en Java.

c) Componente encargado de realizar las operaciones de balanceador.

103. Selecciona la respuesta correcta con respecto al resultado del siguiente bloque de código.

```
public class Test2 extends Thread{
    public static void main(String[] args) {
        protected Thread t = new Thread(new Test2());
        Thread t2 = new Thread(new Test2());

        t.start();
        t2.start();
    }

    public void main(){
        for (int i = 0;i<2;i++)
            System.out.println(Thread.currentThread().getName()+" ");
    }
}
```

La respuesta es: **NO COMPILA**

1. Comando Docker que se utiliza para construir la imagen a partir del Dockerfile

El comando Docker que se utiliza para construir una imagen a partir de un Dockerfile es docker build. Este comando lee el Dockerfile y el contexto de construcción para crear una imagen que se puede usar para ejecutar contenedores.

2. ¿Seleccione la respuesta que considere correcta, dado el siguiente bloque de código?

```
import java.util.Arrays;
```

```
public class Example {
```

```
    public static void main( String [] args ) {  
        int [] [] matrix= {{ { 1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};  
        int [] [] flattened= Arrays.stream(matrix)  
            .flatMapToIn(layer)Arrays.string(layer)  
            .flatMapToIn(Arrays::stream))  
            .boxed().map(n -> new int[(n)).toArray(int[] []::new);
```

```
        System.out.println(Arrays.deepToString(flattened));  
    }  
}
```

- a) [[1], [2], [3], [4], [5], [7], [8]]
- b) [[1, 2]], [[3, 4]], [[5, 6]], [[7, 8]]
- c) **[[1, 2], [3,4], [5,6], [7,8]]**