**Examen Teórico-Práctico de Java para Programadores Senior (Java 11 / Java 17)**

**Nombre:**

**Fecha:**

**Sección 1: Preguntas Teóricas**

1. **¿Cuál es la diferencia entre var y un tipo explícito en Java 11 y Java 17?**
   * Explica cómo se manejan las inferencias de tipo en Java 11 y Java 17 al utilizar var. ¿Hay diferencias significativas entre ambas versiones? Da ejemplos.

La diferencia entre var y un tipo explicito es que en var no declaras el tipo de variable, el compilador lo infiere y el explicito si le indicas el tipo de variable, ejemplo:

var usuario = “Luis”;

String usuario = “Luis”;

En este ejemplo en uno estamos Infiriendo el tipo de dato, es decir no le decimos a java que es un dato de tipo “String” sin embargo java lo infiere y lo hace de tipo string, en el otro le decimos explicitamente que la variable es un String.

La diferencia entre como se utiliza la inferencia de datos en java 11 y java 17 es que en java 11 no se puede utilizar la inferencia de datos en lambdas y en java 17 si.

1. **Explique la diferencia entre un Record y una Class tradicional en Java.**
   * ¿Qué ventajas ofrece un Record en comparación con una clase convencional? ¿En qué escenarios es preferible usar un Record?

Un Record es como una clase pero esta no tiene métodos mas que el constructor, es decir solo almacena datos y no es mutable (no se le puede cambiar de valor una vez creado). A la clase se le puede agregar métodos y si son mutables.

Puedes usar un record cuando solo requieras modelar datos, tiene algunas ventajas como el hacerlo mas rapido y simple ya que con una clase tendrías que hacer los setters, getters y el constructor. Yo lo utilizaría para hacer los request y los response con spring.

1. **¿Qué es un sealed class en Java 17? ¿Cómo se usa y qué ventajas ofrece?**
   * Explica cómo funcionan las clases selladas en Java 17 y proporciona un ejemplo de cómo se pueden utilizar para modelar jerarquías de clases.

Una clase sellada es una clase que no se puede extender, es decir que no permite tener clases hijas, se usa para seguridad para indicar que no se debe de extender porque es la ultima abstracción de la clase que se esta creando.

Por ejemplo:

public sealed interface Mascota permits Gato, Perro {

void comer();

}

public final class Gato implements Mascota {

public void comer() {

System.out.println("Comiendo");

}

}

public final class Perro implements Mascota {

public void comer() {

System.out.println("Comiendo");

}

Aqui se creo una interfaz de tipo “Mascota” y se extendieron de ella clases de tipo “Gato” y “Perro”, en este ejemplo supongamos que hay un sistema que solo permite a perros y gatos no mas mascotas, es por eso que ya no de puede extender mas estas clases ya que solo se permite mascotas de tipo gato y perro.

1. **En Java 11, se introdujo la API HttpClient. Explica sus principales características. ¿Cómo se realiza una solicitud GET en un servidor usando la API HttpClient en Java 11?**

Antes de java 11 se utilizaba “HttpURLConnection” y su principal característica de HttpClient es que es mas fácil de utilizar.

Una solicitud GET se hace asi:

HttpRequest request = HttpRequest

.newBuilder()

.uri(URI.create("https://www.prueba.com/usuario"))

.GET()

.build();

1. **¿Qué mejoras importantes introdujo Java 17 respecto a Java 11 en términos de rendimiento y nuevas características?**
   * Menciona al menos tres mejoras clave que afecten el rendimiento o la experiencia de desarrollo.

En java 17 se agregaron los records, Tambien se agreagaron las clases selladas y se hizo una versión LTS.

1. **En Java, ¿cuál es la diferencia entre HashMap y ConcurrentHashMap?**
   * ¿En qué situaciones utilizarías cada uno de ellos? Explica cómo manejan la concurrencia.

El ConcurrentHashMap es seguro para hilos y HashMap no, y yo lo utilizaria ConcurrentHashMap cuando varios hilos intentan obtener y settear datos en un HashMap.

1. **¿Cómo se implementan y gestionan los Streams en Java 11 y 17?**
   * Explica el concepto de Streams en Java, destacando sus diferencias de uso en Java 11 y 17. Proporciona un ejemplo donde se utilicen operaciones de filtrado, transformación y agrupación de datos.

Stream es para manipular datos de forma declarativa como se haria con SQL en bases de datos, pero aquí se hace con arreglos, esto hace que el codigo sea mas simple y entendible. Las diferencias entre java 11 y java 17 fue que agregaron mas metodos como takeWhile y DropWhile.

Ejemplo:

productos.stream().filter(p -> p.getPrecio() > 50.0); // Filtra un arreglo con objetos de tipo productos en donde el precio sea mayor a 50.

productos.stream().map(p -Z p.getName() ); // Transforma todos los productos a un objeto de tipo String solo con su nombre

Map<Character, List<String>> productosAgrupados = productos.stream().collect(Collectors.groupingBy(s -> s.getName().charAt(0))); // Agrupa todos los elementos del arreglo por su primera letra del nombre de cada producto

1. **Explica cómo funciona el sistema de módulos en Java 9 y su evolución hasta Java 17.**
   * ¿Cómo se maneja el acceso entre módulos? ¿Cómo afecta esto a la estructura de aplicaciones grandes y monolíticas?

El sistema de modulos en java 9 es para hacer referencia a los modulos que vas utilizar en tu proyecto para no utilizar todo y que se vuelv un monolito. Y su evulucion es que en java 17 intrudujeron las flags: “--add-exports” y “—add-opens”. Se utiliza en un archivo “module-info.java” aquí declaras que modulo depende de que otro.

1. **¿Qué es un Pattern Matching en Java 17 y cómo mejora la legibilidad del código?**
   * Describe el concepto y la sintaxis de Pattern Matching en Java 17 y cómo ayuda a reducir el código repetitivo.

El Pattern Matching es para saber si un objeto es de una clase u otra y en java 17 se hicieron mejoras para simplificar el codigo, antes debias de comprobar de que tipo era, declarar la variable y despues hacer el cast, en java 17 se hace todo eso en una solo linea.

Ejemplo:

if (obj instanceof String s) {

System.out.println("Es una cadena de longitud: " + s.length());

}

**Sección 2: Ejercicios Prácticos**

**Ejercicio 1: CRUD de Cliente con Record y Pattern Matching**

**Objetivo:** Implementar un CRUD de clientes utilizando un Record para representar a los clientes y aplicar Pattern Matching para trabajar con sus tipos.

1. **Crea un Record en Java 17** que represente a un **Cliente** con los siguientes atributos: id (String), nombre (String), email (String), edad (int), y tipoCliente (enum que puede ser REGULAR, VIP).
2. **Implementa un CRUD** (Crear, Leer, Actualizar, Eliminar) para gestionar los clientes en memoria:
   * Utiliza una List<Cliente> para almacenar los clientes.
   * Crea métodos para:
     + Crear un nuevo cliente.
     + Obtener todos los clientes.
     + Actualizar los datos de un cliente por su ID (por ejemplo, cambiar su email o su tipo de cliente).
     + Eliminar un cliente por su ID.
3. **Usa Pattern Matching** para realizar una acción específica si el cliente es de tipo VIP o REGULAR. Por ejemplo, un cliente VIP tiene un descuento en los préstamos, mientras que un cliente REGULAR no.

**Ejercicio 2: CRUD de Préstamos y Manejo de Intereses**

**Objetivo:** Implementar un CRUD para gestionar préstamos de clientes y aplicar diferentes tasas de interés según el tipo de cliente.

1. **Crea una clase Prestamo** con los siguientes atributos:
   * id (String), monto (double), clienteId (String), fecha (LocalDate), estado (enum con valores PENDIENTE, PAGADO).
2. **Implementa un CRUD** para gestionar los préstamos:
   * Crear un préstamo.
   * Obtener todos los préstamos activos.
   * Actualizar el estado de un préstamo (por ejemplo, cambiar de PENDIENTE a PAGADO).
   * Eliminar un préstamo.
3. **Calcula el interés**:
   * Los clientes VIP tendrán una tasa de interés del 5% y los REGULAR tendrán una tasa del 10%.
   * Implementa un método que calcule el monto total a pagar (monto + interés) en función del tipo de cliente.

Tomando en cuenta el planteamiento anterior, considerar los siguientes puntos.

* Agregar contrato de interfaz yaml con estándar OpenAPI 2.0.
* Pruebas Unitarias (mínimo un método/caso).
* Legibilidad del código.
* Implementación de logs