**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.

En ambas versiones var se se comporta de la misma manera esencial.

La inferencia ocurre en tiempo de compilación, nunca en tiempo de ejecución. El compilador determina el tipo exacto basado en el valor de la derecha del “ = ”, y luego lo reemplaza internamente como si lo hubieras escrito

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?

Record y Class tradicional tienen varias diferencias, pero las principales son

Que Record por defecto es inmutable, mientras que la clase tradicional es mutable o inmutale.

Otra característica es que los métodos generados en record son automáticos y en una clase tradicional se deben escribir manualmente

Es preferible usar record cuando necesitas garantizar la inmutabilidad y no necesitas lógica compleja en métodos

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.

Las clases selladas son una forma de restringir que clases pueden extender o implementar una clase o interfaz. Las subclases deben ser final, sealed o non-sealed

Un ejemplo: public sealed class Animal permits Perro, Gato{}

public final class Perro extends Animal{}

public final class Gato extends Animal{}

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?**

**Una de las características es que ya venia incluido en el JDK, no se necesitaba ninguna biblioteca externa.**

**Esta orientado a objetos como HttpRequest, HttpResponse y HttpClient**

**Para hacer una solicitud GET a un servidor utilizando la API se necesitan 3 pasos:**

**1.- crear un cliente HTTP**

**2.-Construir la solicitud HTTP**

**3.-Enviar la solicitud y manejar la respuesta**

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.

Mejor rendimiento de la JVM y recolección de basura mas eficiente

Introdujo Record, pattern matching y sealed classes que simplifican el codigo

Herramientas modernas que mejoran las mantenibilidad

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

Tanto HashMap como ConcurrentHashMap son implementaciones de la interfaz Map, pero su comportamiento en entornos concurrentes es muy diferente

Se usa HashMap si solo un hilo accede al mapa

Se requiere un rendimiento máximo sin necesidad de sincronización

ConcurrentHashMap esta diseñado específicamente para entornos concurrentes

Se usa si quieres evitar sincronización manual

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.

Los Streams son una API para procesar colecciones de datos de forma declarativa y funcional. No hay cambios radicales en la API de Streams entre java 11 y java 17, pero hay pequeñas mejoras

Métodos como Predicate.not(…) para filtros mas legibles en java 11

Mejoras en pattern Matching y collectors.teeing() para combinaciones de resultados

public class EjemploStream {

record Persona(String nombre, int edad, String ciudad) {}

public static void main(String[] args) {

List<Persona> personas = List.of(

new Persona("Ana", 20, "Lima"),

new Persona("Luis", 17, "Lima"),

new Persona("Marta", 25, "Bogotá"),

new Persona("Carlos", 30, "Lima"),

new Persona("Laura", 16, "Bogotá")

);

Map<String, List<String>> resultado = personas.stream()

.filter(p -> p.edad() >= 18) // 1. Filtrar adultos

.map(p -> new Persona(p.nombre().toUpperCase(), p.edad(), p.ciudad()))

.collect(groupingBy(

Persona::ciudad,

mapping(Persona::nombre, toList())

));

System.out.println(resultado);

}

}

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 módulos tiene como función mejorar la modularidad, seguridad y mantenimiento del código java, especialmente en aplicaciones grandes y monolitas

Declaracion en module-info.java exports paquete permite que otros módulos accedan a ese paquete

Requires nombreModulo declara dependencia a otro modulo

Opens paquete Permite acceso por reflexión

Encapsulamiento fuerte: solo las clases exported son visibles, lo que ayuda a evitar uso indebido de clases internas.

Mejor mantenimiento: dependencias claras y controladas.

Carga más rápida: el sistema puede evitar cargar clases innecesarias.

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.

Pattern Matching es una caracteristica del lenguaje que simplifica las operaciones comunes de verificación de tipos y extracción de datos.

Antes era necesario hacer verificación de tipo + cast manual, de esta manera el patron es repetitivo y propenso a errores. Con java 17 puedes combinar verificación de tipo y declaración de variable en una sola expresión

**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