Instituto Tecnológico del Putumayo (ITP) Asignatura: Programación Orientada a Objetos Semestre: Tercero Proyecto Final del Semestre - Desarrollo de API Web con Java

Fecha: 8 de abril de 2025

Estimados Estudiantes,

El propósito de este proyecto es aplicar de manera práctica los conceptos fundamentales de la Programación Orientada a Objetos (POO) aprendidos durante el semestre en un contexto de desarrollo de software moderno: la creación de una API web.

1. Introducción: Elección de Tecnologías

- **¿Por qué Java?** Hemos seleccionado Java como el lenguaje principal para este proyecto por varias razones clave. Java es un lenguaje robusto, maduro y ampliamente utilizado en la industria, especialmente en el desarrollo de aplicaciones empresariales a gran escala. Su diseño está intrínsecamente ligado a los principios de la POO (Encapsulamiento, Herencia, Polimorfismo, Abstracción), lo que lo convierte en una excelente herramienta para aprender y reforzar estos conceptos. Su fuerte tipado y manejo de excepciones promueven la escritura de código más seguro y mantenible. Además, su vasta comunidad y ecosistema de librerías y frameworks ofrecen un gran soporte para el desarrollo.
- **¿Por qué SparkJava (o un Framework similar)?** Para la construcción de la API web, utilizaremos principalmente el micro-framework **SparkJava**.
-
```
<dependency>
    <groupId>com.sparkjava</groupId>
    <artifactId>spark-core</artifactId>
    <version>2.9.4</version>
</dependency>
```

  La elección de SparkJava se basa en su **simplicidad y flexibilidad**. Es un framework minimalista que permite crear APIs web rápidamente con muy poco código "boilerplate" (código repetitivo de configuración). Esto nos permite enfocarnos en la lógica de negocio, el diseño de las clases (Modelos) y la implementación de los principios POO, sin vernos abrumados por la complejidad de frameworks más grandes. Su naturaleza ligera lo hace ideal para aprender los fundamentos de las APIs RESTful.

  Alternativa: Como se menciona más adelante, el uso de **Spring Boot** es permitido. Si bien es un framework más complejo y robusto, ofrece muchas características "out-of-the-box" y es el estándar de facto en muchas empresas. Si eligen Spring Boot, deberán justificar por qué consideran que es una mejor opción para su proyecto y nivel de aprendizaje actual.

2. Objetivo General del Proyecto

Desarrollar una aplicación web backend (API RESTful) en Java que gestione la información de un dominio de problema elegido por el grupo (ej: inventario de tienda, gestión de biblioteca, sistema de reservas simple, etc.), aplicando correctamente los conceptos de POO y siguiendo buenas prácticas de codificación.

3. Especificaciones Técnicas

- **Lenguaje:** Java (versión 8 o superior).
- **Framework Web:** SparkJava (recomendado) o Spring Boot (permitido con justificación).
- **Gestor de Dependencias:** Maven o Gradle.
- **Persistencia: No se requiere base de datos.** La persistencia de datos se simulará utilizando clases repositorio en memoria.

Ejemplo:

```java
import java.util.ArrayList;
import java.util.Collections; // Buena práctica para devolver copias
import java.util.List;
import java.util.Optional;
import java.util.concurrent.atomic.AtomicInteger; // Para IDs thread-safe

// Ejemplo para una clase 'Producto'
public class ProductosRepositorio {
    // Usamos List en lugar de ArrayList para mayor flexibilidad
    private static final List<Producto> productos =
Collections.synchronizedList(new ArrayList<>());
    // Usamos AtomicInteger para generar IDs únicos de forma segura en
concurrencia
    private static final AtomicInteger autoId = new AtomicInteger(1);

    // Método para obtener todos los productos (devuelve una copia inmutable)
    public static List<Producto> obtenerTodos() {
        return new ArrayList<>(productos); // Devuelve una copia para evitar
modificaciones externas
    }

    // Método para obtener un producto por ID
    public static Optional<Producto> obtenerPorId(int id) {
        return productos.stream()
                .filter(p -> p.getId() == id)
                .findFirst();
    }

    // Método para agregar un nuevo producto
    public static Producto agregar(Producto producto) {
        producto.setId(autoId.getAndIncrement()); // Asigna y luego
incrementa el ID
```

```
            productos.add(producto);
            return producto;
        }

        // Método para actualizar un producto existente
        public static Optional<Producto> actualizar(int id, Producto
    productoActualizado) {
            Optional<Producto> productoExistenteOpt = obtenerPorId(id);
            if (productoExistenteOpt.isPresent()) {
                Producto productoExistente = productoExistenteOpt.get();
                // Actualizar los campos necesarios (ejemplo)
                productoExistente.setNombre(productoActualizado.getNombre());
                productoExistente.setPrecio(productoActualizado.getPrecio());
                // ... otros campos
                return Optional.of(productoExistente);
            }
            return Optional.empty(); // No encontrado
        }

        // Método para eliminar un producto por ID
        public static boolean eliminar(int id) {
            return productos.removeIf(p -> p.getId() == id);
        }
    }
```

- **Servidor:** El servidor embebido proporcionado por SparkJava o Spring Boot (Tomcat, Jetty, Undertow).

4. Requisitos Clave (Para la Entrega Final)

1. **Modelos (Clases):** El proyecto debe contener un mínimo de **20 clases de modelo** que representen las entidades del dominio elegido. Estas clases deben estar bien diseñadas, aplicando encapsulamiento (atributos privados, métodos getters/setters o constructores apropiados).
2. **Operaciones CRUD:** Cada clase de modelo debe tener implementadas las operaciones básicas de **C**reate (Crear), **R**ead (Leer - obtener todos y obtener por ID), **U**pdate (Actualizar) y **D**elete (Eliminar). Estas operaciones deben ser expuestas a través de endpoints de la API RESTful.
3. **Código Limpio y Organización:**
    - Evitar definir **todas** las rutas (endpoints) directamente en la clase `Main`. Se debe buscar una estructura organizada, por ejemplo, creando clases "Controladoras" o "Handlers" separadas para cada recurso/modelo, o utilizando mecanismos de agrupación de rutas que ofrecen los frameworks.
    - Aplicar principios básicos de código limpio (nombres significativos, métodos cortos y enfocados, evitar duplicación de código).

4. **Control de Versiones (Git y GitHub):**
    - El proyecto completo debe estar alojado en un repositorio de **GitHub**.
    - Se debe evidenciar la **contribución de cada miembro del equipo** a través de los *commits*. Los mensajes de commit deben ser descriptivos.
5. **Documentación (Informe Final):** Un informe final que detalle:
    - Descripción del proyecto y dominio elegido.
    - Diagrama de clases (UML).
    - **Evidencia explícita de cómo se aplicaron los conceptos clave de POO** (Encapsulamiento, Herencia (si aplica), Polimorfismo (si aplica), Abstracción) en el diseño y código.
    - Justificación del framework elegido (si fue Spring Boot).
    - Manual técnico básico (cómo compilar y ejecutar).
    - Conclusiones.

5. Requisitos Específicos para la Primera Entrega (Corte Actual)

Para la primera entrega (fecha por definir), se debe presentar lo siguiente:

1. **Configuración del Proyecto:** El proyecto base debe estar correctamente configurado (usando Maven o Gradle) con las dependencias necesarias (SparkJava o Spring Boot) y debe ser compilable y ejecutable.
2. **Modelos y CRUD Iniciales:** Implementación de al menos **6 clases de modelo** distintas, cada una con sus respectivos **endpoints CRUD funcionales** (Crear, Leer todos, Leer por ID, Actualizar, Eliminar) utilizando el repositorio en memoria.
3. **Repositorio GitHub:** El repositorio en GitHub debe estar creado y contener el código de esta primera entrega, con commits iniciales que muestren el progreso.
4. **Informe de Pruebas:**
    - Un documento formal (utilizando el **formato oficial del ITP**) que evidencie las pruebas realizadas a **todos** los endpoints de la API creados hasta el momento.
    - Se deben utilizar herramientas como **Postman**, **Insomnia** o **Talend API Tester** (extensión de Chrome).
    - El informe debe incluir:
        - Descripción de cada endpoint probado (Método HTTP, URL, propósito).
        - Capturas de pantalla de la herramienta de pruebas mostrando la petición (Request) y la respuesta (Response) para cada operación CRUD de cada modelo (tanto casos exitosos como, opcionalmente, casos de error básicos, ej: ID no encontrado).
        - Breve descripción del resultado de cada prueba (éxito/fallo).
    - Si se elige **Spring Boot**, este informe debe incluir la **justificación** de por qué se seleccionó este framework sobre SparkJava.

6. Rúbrica de Calificación - Primera Entrega

Criterio	Descripción	Porcentaje
1. Configuración del Proyecto	El proyecto está correctamente configurado (Maven/Gradle), las dependencias están incluidas y el proyecto compila y se ejecuta sin errores.	10%
2. Implementación de Modelos (Mínimo 6)	Se han creado al menos 6 clases de modelo distintas, con atributos privados y métodos de acceso apropiados (encapsulamiento básico).	15%
3. Implementación de CRUD	Funcionalidad CRUD completa (Crear, Leer todos, Leer por ID, Actualizar, Eliminar) implementada para los 6 modelos usando repositorios en memoria.	25%
4. Funcionalidad y Pruebas de API	Los endpoints de la API para los 6 modelos responden correctamente a las peticiones según lo esperado para cada operación CRUD.	20%
5. Organización Inicial del Código	Se evidencia un esfuerzo inicial por no colocar toda la lógica de rutas en `Main` (ej. uso de clases separadas, métodos helper).	10%
6. Informe de Pruebas y Justificación	El informe sigue el formato ITP, es claro, incluye capturas de pantalla de pruebas para *todos* los endpoints y resultados. Si aplica, incluye la justificación de Spring Boot.	5%
7. Repositorio GitHub y Commits Iniciales	El repositorio existe en GitHub, contiene el código de la entrega y muestra commits iniciales del equipo.	15%
TOTAL		**100%**

Notas Finales:

- Este proyecto es una excelente oportunidad para consolidar sus conocimientos de POO y adquirir experiencia en el desarrollo de APIs, una habilidad muy demandada en la industria.
- Fomenten el trabajo en equipo, la comunicación y la aplicación de buenas prácticas desde el inicio.
- No duden en consultar cualquier duda.