Desarrollo de Software
Prof. Carlos Alonzo
Tercer Examen Parcial
Enero 2024

Nombre: _____

El examen es individual. Lea con detenimiento y tranquilidad cada uno de los enunciados. Justifique todas sus respuestas.

(4 puntos) Pregunta 1

La taquilla del Estadio Monumental de Caracas invita a empresas de software a presentar ofertas para un proyecto. Quieren tener un sistema o software (con clases e interfaces) para calcular el precio de venta de entradas. Estas son las reglas para la venta de entradas en el Estadio:

1. El precio de venta de entradas consta de dos partes: precio de entrada e impuesto de venta
2. El precio del boleto varía entre niños (menores de 16 años a $10), adultos ($15) y personas mayores (mayores de 65 años a $12)

La taquilla del estadio adjudicará el proyecto al diseño que permita agregar más categorías de entradas y cambios en el cálculo de la tasa impositiva fácilmente.

Debe presentar su diseño utilizando el patrón Estrategia (*Strategy*), construyendo su **Diagrama de clases UML**, que modele el problema planteado.

(4 puntos) Pregunta 2

Una clase de Oráculo expone un método para devolver un número aleatorio (`printNumber`). Se le pide, con el paso del tiempo, ampliar el comportamiento de la clase Oráculo para permitirle:

1. imprimir un mensaje de bienvenida antes de buscar el número
2. imprimir un mensaje de saludo al final
3. imprimir los dos mensajes anteriores, incluso en diferente orden

Usted debe indicar con precisión **cuál patrón de diseño** usaría para resolver este problema. Debe argumentar su respuesta construyendo el **Diagrama de clases UML**, donde se evidencie con claridad el patrón de diseño que permite cambiar el comportamiento en tiempo de ejecución (*runtime*) de la clase Oráculo.

(3 puntos) Pregunta 3

Lee con detenimiento el siguiente código fuente:

```
class Result<T> {

    resultado: T;

    error?: Error;

    isError: boolean = false;

    constructor (r: T) {

       this.resultado = r;

    }

    setResultado(r: T) { this.resultado = r; }

    setError(e: Error) { this.error = e; }

}

interface CursosRepositorio {

     getSuscriptores(cursoId: number): number[];

}

interface ISenderPush {

     send(userId: number): void;

}

interface IServicio<TComando, TResultado> {

     execute(c: TComando): Result<TResultado>;

}
```

En el código fuente existe un *code smell* por la ausencia del uso de un **patrón táctico de DDD**. Indique cuál es ese patrón y dónde debe aplicarlo. Justifique su respuesta.

(5 puntos) Pregunta 4

Considere el servicio que se encarga de aplicar un cupón sobre una suscripción, es decir, el servicio que se encarga de aplicar un cupón válido al tiempo de suscripción de un usuario. Cuando se aplica el cupón, se extiende el tiempo de vigencia de una suscripción de un usuario. Se describen a continuación los pasos de implementación de este caso de uso o funcionalidad del sistema:

1. Existe un controlador web en la capa de infraestructura que recibe como parámetros el identificador del cupón (**`idCupon`**) y el identificador de la suscripción de un usuario (**`idSuscripcion`**)
2. El controlador web funciona como el *endpoint* de un API REST, y retorna un status code 200 en caso de éxito (es decir, que se aplicó el cupón exitosamente a la suscripción). Pero puede retornar un status code 503 en el caso que el cupón sea inválido (no existe, ya fue aplicado o ha expirado) y puede retornar un status code 504 en el caso de que la suscripción es inválida (no existe, está bloqueada o cancelada)
3. Es importante tener en cuenta que se debe llevar un log de auditoria cada vez que se aplica un cupón. Si la aplicación del cupón es éxito, se registra en una base de datos especial (una base de datos en *FireStore* que se basa en colecciones). Y cuando se produce un error se registra en la misma base de datos, pero en otra colección, donde siempre se lleva registro del tipo de error.
4. Es muy importante que una vez que se aplica con éxito un cupón, el mismo ya no es válido. Es decir, se debe marcar o cambiar su estado como usado, puesto que ya fue aplicado a una suscripción.

Usted debe hacer su diseño utilizando todos los patrones, técnicas y buenas prácticas vistas en clases: patrones tácticos de DDD, Programación Genérica, Programación Orientada a Aspectos, Arquitectura Hexagonal y los Principios SOLID.

Usted debe presentar su diseño en un diagrama de clases UML enriquecido con los estereotipos DDD, especificando con claridad los métodos y constructores de sus clases y destacando con los colores respectivos en cuál capa de la Arquitectura Hexagonal se encuentra cada abstracción. También su código debe usar los genéricos **`Optional`** y **`Result`** (o **`Either`**, según usted desee) para el manejo de las excepciones.

IMPORTANTE: Modele las abstracciones que son necesarias para el caso de uso que se ha descrito. No agregue objetos innecesarios a su diseño.

(2 puntos) Pregunta 5

Suponga que usted está diseñando la capa de dominio utilizando DDD para un sistema de facturación. Este sistema maneja varios conceptos importantes: presupuestos, facturas, clientes, impuesto, moneda, tasa de cambio, entre otros. Adicionalmente, las tasas de cambio se consultas en servicios (API) que son externos. De igual manera los impuestos también se calculan en un

servicio externo. Es decir, para calcular una tasa de cambio entre monedas o para calcular impuestos, se debe consumir APIs que son externas a nuestro sistema.

¿Cuál sería el **patrón táctico de DDD** que usted debe utilizar para modelar en la capa de dominio la tasa de cambio y los impuestos y sus respectivos cálculos cumpliendo la **Regla de Dependencia de la Arquitectura Hexagonal**? Justifique con claridad cómo lo implementaría.

(2 puntos) Pregunta 6

Justifique cómo usted mejoraría la ***testability*** de la siguiente clase, considerando los principios vistos en clases y planteados por ***Maurício Aniche*** en su libro ***Effective Software Testing***.

```
public class OrderDeliveryBatch {

	public void runBatch() {

		OrderDao dao = new OrderDao();

		DeliveryStartProcess delivery = new DeliveryStartProcess();

		List<Order> orders = dao.paidButNotDelivered();

		for (Order order: orders) {

			delivery.start(order);

			if (order.isInternational()) {

				order.setDeliveryDate("5 days from now");

			} else {

				order.setDeliveryDate("2 days from now");

			}

		}

	}

}

class OrderDao {// accede a la base de datos}

class DeliveryStartProcess {// se comunica con un servicio externo}
```