Desarrollo de Software
Prof. Carlos Alonzo
Tercer Examen Parcial
04-07-2024 **Nombre:** _____

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

1 Verdadero y falso (4 puntos)

A continuación se enuncian varias afirmaciones, usted debe indicar si es Verdadera o si es Falsa. **Se aplica factor de corrección: una respuesta incorrecta anula una buena**.

1. Los servicios de dominio encapsulan procesos o políticas del negocio, o también representan algún contrato a nivel de la lógica de negocio. Ellos representan algún comportamiento, no tienen identidad, son stateless, con frecuencia orquestan con múltiples entidades, objetos del dominio y mediante puertos orquestan algunas responsabilidades con capa de aplicación.
   ☐ Verdadero ☐ Falso

2. Un modelo de dominio anémico se caracteriza por tener un conjunto de servicios de dominio que capturan toda la lógica del dominio, realizan todos los cálculos y actualizan los objetos del modelo. Estos servicios se encuentran encima del modelo de dominio y utilizan el modelo de dominio para los datos.
   ☐ Verdadero ☐ Falso

3. Cuando un proceso o transformación importante en el dominio no es responsabilidad natural de un agregado o value object, incorpore una operación al modelo como una interfaz independiente declarada como un servicio, definiendo con claridad el contrato de dicho servicio de aplicación.
   ☐ Verdadero ☐ Falso

4. Las entidades se encargan de manejar su propio ciclo de vida, ya que son objetos mutables y manejan sus estados.
   ☐ Verdadero ☐ Falso

5. La inmutabilidad, la cohesión y la combinabilidad son las tres cualidades de los value objects que los hacen difíciles de probar, afectando así su testeability.
   ☐ Verdadero ☐ Falso

6. La creación de objetos mediante las fábricas no es una responsabilidad de la capa de dominio, pero por convención reside dentro de la capa de dominio. Sin embargo, para reconstruir un objeto a partir de un DTO, se hace mediante un Factory Method en la capa de aplicación.
   ☐ Verdadero ☐ Falso

7. Millett plantea en su libro, la existencia de dos tipos de eventos de dominio: internos y externos. En el caso de los eventos internos, estos se pueden componer de otros objetos de la capa de dominio.
   ☐ Verdadero ☐ Falso

8. Ningún objeto de dominio fuera del límite del agregado puede contener una referencia a objetos internos. El Aggregate Root puede devolver copias de objetos internos para su uso en operaciones de dominio, pero cualquier cambio en estos objetos debe realizarse a través del Aggregate Root.
   ☐ Verdadero ☐ Falso

2 Completación (2 puntos)

Complete los siguientes enunciados:

1. La capa de aplicación en la Arquitectura Hexagonal está libre de código de infraestructura. Las principales abstracciones que se encuentran en esta capa son los _____ _____ y los _____.

2. Se considera una buena práctica mantener todos los getters/setters de las entidades como privados, para proteger el estado y mantenerlo encapsulado del resto de la aplicación. Se recomienda usar el patrón de diseño _____ para evitar exponer el estado de las entidades.

3. Las capas en la Arquitectura Hexagonal sólo deben tener dependencias con abstracciones de las capas internas. Esa afirmación se conoce como la _____ _____.

4. El repositorio actúa como un mediador entre el agregado y el modelo de persistencia (data model). Si no queremos exponer los atributos del agregado (su estado) y deseamos que estén totalmente encapsuladas, el repositorio se puede implementar utilizando el patrón de diseño _____.

Desarrollo de Software
Tercer Examen Parcial

3 Ejercicio: DDD, Arquitectura Hexagonal y AOP (7 puntos)

Usted fue contratado para implementar la siguiente funcionalidad de un sistema de gestión de tareas, haciendo uso de sus conocimientos de DDD, AOP (aspectos) y Arquitectura Hexagonal. **Usted debe programar en TypeScript un (1) servicio de dominio, un (1) de servicio de aplicación y los dos (2) aspectos que modelen el problema**. Usted debe identificarlos luego de leer el planteamiento del problema con mucho detenimiento. **Usted también debe programar la línea de instanciación del servicio construyendo el Composition Root al detalle**.

Planteamiento del problema: Existe una funcionalidad para asignar una tarea a un developer, esta asignación la hace el sistema sin intervención humana. La tarea dispone de un tiempo estimado de realización y un estado: $ToDo$, $InProgress$, $Complete$. Un developer puede tener diferentes rangos: $Junior$, $SemiSenior$, $Senior$. Cada developer dispone de 146 horas al mes disponibles para hacer tareas. Cuando se desea asignar una tarea a un developer, se buscan todos los developers que tengan horas disponibles suficientes en el mes para poder ejecutar la tarea. Este proceso se implementa como una revisión secuencial de los developers con horas disponibles: si se tiene un developer $Senior$ o $SemiSenior$ con tiempo disponible, se le asigna la tarea, siempre que le quede un margen de 24 horas de holgura en el mes. Si es un developer $Junior$, se le asigna la tarea si la suma del tiempo de sus tareas asignadas (en estado $ToDo$ o $InProgress$,) es menor o igual a 100 horas. Si no se consigue asignar la tarea a un developer, se lanza una excepción de dominio ($AssignTaskToDeveloperError$).

Se debe tener un log donde se registre la tarea y a quien quedó asignada, y también un log de los errores/excepciones que se puedan producir en el servicio de aplicación. Adicionalmente, se desea llevar un registro del performance o desempeño del servicio de aplicación, es decir, cuántos milisegundos se tarda ejecutar el servicio de asignar la tarea a un developer. Debe usar las abstracciones usadas en clases, a continuación se presentan algunas de ellas a las cuales se debe apegar:

```
interface IService<TService, RService> {
  execute(s: TService): Either<Error, RService>;
}

class Optional<T> {
  hasValue(): boolean { ... }
  getValue(): T { ... }
}

class Either<TLeft, TRight> {
  isLeft(): boolean { ... }
  getLeft(): TLeft { ... }
  isRight(): boolean { ... }
  getRight(): TRight { ... }
  static makeLeft<TLeft, TRight>(value: TLeft) { ... }
  static makeRight<TLeft, TRight>(value: TRight) { ... }
}
```

4 Ejercicio: Modelo de dominio con Domain-driven Design (DDD) (7 puntos)

Planteamiento del problema: Usted fue contratado para implementar el sistema una empresa que hace ventas *online* de productos relacionados a sistemas de vigilancia o sistemas de control de acceso. Se define un (1) **Bounded Context** denominado Ventas (*Sales*) que posee tres (3) **Módulos** gestionados por equipos de developers diferentes: módulo de Producto (*Product*), módulo de Inventario (*Stock*) y módulo Compras (*Purchase*).

Un producto tiene un nombre, un precio (conformado por monto y moneda), una descripción, imágenes del productos y manuales en formato pdf del producto. Se debe llevar un histórico de los precios del producto y también se lleva un control de versiones de los manuales. Una compra contiene la fecha, la lista ordenada de productos con la cantidad y precio del momento de la compra y el método de pago. El inventario posee el id del producto, código único comercial SKU, nombre y cantidad del producto en *stock*. Cuando se produce una compra exitosa, se debe decrementar el inventario. El proceso de *checkout* consiste en sumar el precio de los productos que se van a comprar, se aplica un descuento si hay alguna promoción activa, se calculan los impuestos (*taxes*) y se efectua el pago. Para el cálculo de los impuestos (en porcentaje) se hace uso de un servicio externo en base al país y jurisdicción (municipio). De igual forma los métodos de pago pueden ser dos: tarjetas de crédito (nombre, número

de tarjeta, fecha vencimiento y CVC/código de seguridad) o por PayPal (usuario vía correo registrado en PayPal). Se estima tener otros medios de pago a futuro. Se puede pagar en bolívares o dólares por ello es importante usar el API externo de *Exchange rate lookup* para calcular la tasa de cambio. Se debe guardar si la compra se hizo en Bs. o dólares. Debe tener presente que se debe hacer un uso adecuado del manejo de excepciones de dominio cuando se producen errores en servicios externos.

Debe elaborar el Diagrama de Clases UML del modelo de dominio, enriquecido con los estereotipos DDD para identificar los patrones tácticos, definiendo con claridad las fronteras de los agregados. Debe mostrar todas las relaciones entre los objetos del modelo de dominio. De igual modo recuerde que debe especificar con claridad en pseudo-código los métodos/comportamientos o responsabilidades de todos los agregados y servicios de dominio (explicar en pseudocódigo qué hacen esos métodos relevantes). Debe modelar solo los eventos de dominio y excepciones de dominio que sean relevantes para lo que se describe en este enunciado, es decir, ahorre espacio en su modelo de dominio dejando de modelar todo aquello que no se expone en lo descrito en el planteamiento del problema. **Su respuesta se considera INVÁLIDA si NO especifica los métodos con sus parámetros y valores de retorno y sus tipos**.