## Desarrollo de Software Prof. Carlos Alonzo Tercer Examen Parcial

**04-07-2024 Nombre:** JESÚS PÉREZ

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

# Verdadero y falso (4 puntos)

A continuaci´on se enuncian varias afirmaciones, usted debe indicar si es Verdadera o si es Falsa. **Se aplica factor de correccio´n: una respuesta incorrecta anula una buena**.

* 1. Los servicios de dominio encapsulan procesos o pol´ıticas del negocio, o tambi´en rep- resentan algu´n contrato a nivel de la l´ogica de negocio. Ellos representan algu´n com- portamiento, no tienen identidad, son stateless, con frecuencia orquestan con mu´ltiples entidades, objetos del dominio y mediante puertos orquestan algunas responsabilidades con capa de aplicaci´on. VERDADERO
  2. Un modelo de dominio an´emico se caracteriza por tener un conjunto de servicios de dominio que capturan toda la l´ogica del dominio, realizan todos los c´alculos 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
  3. Cuando un proceso o transformaci´on importante en el dominio no es responsabilidad natural de un agregado o value object, incorpore una operaci´on al modelo como una interfaz independiente declarada como un servicio, definiendo con claridad el contrato de dicho servicio de aplicaci´on. VERDADERO
  4. Las entidades se encargan de manejar su propio ciclo de vida, ya que son objetos mutables y manejan sus estados. FALSO
  5. La inmutabilidad, la cohesi´on y la combinabilidad son las tres cualidades de los value objects que los hacen difíciles de probar, afectando así su testeability. FALSO
  6. La creaci´on de objetos mediante las f´abricas no es una responsabilidad de la capa de dominio, pero por convenci´on 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 aplicaci´on. 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
  8. Ningu´n objeto de dominio fuera del l´ımite 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 trav´es del Aggregate Root. VERDADERO

# Completaci´on (2 puntos)

Complete los siguientes enunciados:

* 1. La capa de aplicaci´on en la Arquitectura Hexagonal est´a libre de c´odigo de infraestruc- tura. Las principales abstracciones que se encuentran en esta capa son los casos de uso y los Servicios de aplicación (orquestadores).
  2. Se considera una buena pr´actica mantener todos los getters/setters de las entidades como privados, para proteger el estado y mantenerlo encapsulado del resto de la apli- caci´on. Se recomienda usar el patr´on de disen˜o Factory Method para evitar exponer el estado de las entidades.
  3. Las capas en la Arquitectura Hexagonal s´olo deben tener dependencias con abstrac- ciones de las capas internas. Esa afirmacion se conoce como el Principio de Inversión de Dependencia.
  4. El repositorio actu´a como un mediador entre el agregado y el modelo de persisten- cia (data model). Si no queremos exponer los atributos del agregado (su estado) y deseamos que est´en totalmente encapsuladas, el repositorio se puede implementar uti- lizando el patr´on de disen˜o repository.

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

Usted fue contratado para implementar la siguiente funcionalidad de un sistema de gesti´on de tareas, haciendo uso de sus conocimientos de DDD, AOP (aspectos) y Arquitectura Hexag- onal. **Usted debe programar en TypeScript un (1) servicio de dominio, un (1) de servicio de aplicaci´on y los dos (2) aspectos que modelen el problema**. Usted debe identificarlos luego de leer el planteamiento del problema con mucho detenimiento. **Usted tambi´en debe programar la l´ınea de instanciaci´on del servicio construyendo el Composition Root al detalle**.

**Planteamiento del problema**: Existe una funcionalidad para asignar una tarea a un developer, esta asignaci´on la hace el sistema sin intervenci´on humana. La tarea dispone de un tiempo estimado de realizaci´on 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 revisi´on 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 excepci´on de dominio (*AssignTaskToDeveloperError*).

Se debe tener un log donde se registre la tarea y a quien qued´o asignada, y tambi´en un log de los errores/excepciones que se puedan producir en el servicio de aplicaci´on. Adicionalmente, se desea llevar un registro del performance o desempen˜o del servicio de aplicaci´on, es decir, cu´antos milisegundos se tarda ejecutar el servicio de asignar la tarea a un developer. Debe usar las abstracciones usadas en clases, a continuaci´on 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) { ... }

}

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

**Planteamiento del problema**: Usted fue contratado para implementar el sistema una em- presa 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) **M´odulos** gestionados por equipos de developers diferentes: m´odulo de Pro- ducto (*Product* ), m´odulo de Inventario (*Stock* ) y m´odulo Compras (*Purchase*).

Un producto tiene un nombre, un precio (conformado por monto y moneda), una descripci´on, im´agenes del productos y manuales en formato pdf del producto. Se debe llevar un hist´orico de los precios del producto y tambi´en 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 mo- mento de la compra y el m´etodo de pago. El inventario posee el id del producto, c´odigo u´nico 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 pre- cio de los productos que se van a comprar, se aplica un descuento si hay alguna promoci´on activa, se calculan los impuestos (*taxes*) y se efectua el pago. Para el c´alculo de los impuestos (en porcentaje) se hace uso de un servicio externo en base al pa´ıs y jurisdicci´on (municipio). De igual forma los m´etodos de pago pueden ser dos: tarjetas de cr´edito (nombre, nu´mero

de tarjeta, fecha vencimiento y CVC/c´odigo de seguridad) o por PayPal (usuario v´ıa correo registrado en PayPal). Se estima tener otros medios de pago a futuro. Se puede pagar en bol´ıvares o d´olares 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 d´olares. 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 t´acticos, definiendo con claridad las fron- teras 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-c´odigo los m´etodos/comportamientos o responsabilidades de todos los agregados y servicios de do- minio (explicar en pseudoc´odigo qu´e hacen esos m´etodos 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 INVA´LIDA si NO especifica los m´etodos con sus par´ametros y valores de retorno y sus tipos**.