DSS - Examen teórico - Junio 2016

1 de junio de 2016

- Duración máxima 90 minutos
- Cada respuesta incorrecta resta 1/2 respuesta correcta
- No olvides indicar la modalidad del examen en la hoja de respuestas

Modalidad 3

1. En una arquitectura en capas abierta...
   (a) se permite que las capas inferiores se comuniquen con las superiores.
   (b) las capas superiores pueden saltarse algunas de las capas inferiores.
   (c) se define una capa de servicio que puede ser usada opcionalmente por la capa de lógica de negocio.

2. ¿Con qué patrón podemos mantener la consistencia cuando se modifican dos instancias de un mismo objeto desde distintas partes del sistema?
   (a) Unit of Work
   (b) Identity Map
   (c) Lazy Load

3. ¿En qué capa se deben situar las operaciones complejas de una aplicación?
   (a) En la capa de dominio.
   (b) En la capa de servicio.
   (c) En la capa de presentación.

4. ¿Cuál es la diferencia entre los patrones Table Module y Domain Model?
   (a) Table Module pertenece a la capa de acceso a datos, y Domain Model a la capa de lógica de dominio.
   (b) Normalmente, con Table Module hay un objeto por cada tabla, mientras que con Domain Model hay un objeto por cada fila de la tabla.
   (c) Las dos son ciertas.

5. ¿Qué patrón es más recomendable para añadir nuevas funcionalidades a un objeto?
   (a) Decorator
   (b) Composite
   (c) Adapter

6. ¿Qué patrón permite reutilizar un mismo objeto de acceso a datos para distintas vistas?
   (a) Model View Controller
   (b) Model View Presenter
   (c) Code-behind

7. ¿Cuándo conviene aplicar el patrón GRASP 'Experto en Información' en cascada?
   (a) Cuando queremos prevenir la aparición de nuevas dependencias entre clases.
   (b) Cuando queremos romper intencionadamente la encapsulación de información.
   (c) Cuando queremos traspasar datos de la lógica de negocio a la capa de acceso a datos.

8. ¿Qué tipo de interfaces son deseables para llamadas entre objetos distribuidos?
   (a) Interfaces sin estado.
   (b) Interfaces de grano fino.
   (c) Interfaces de grano grueso.

9. En el patrón GOF 'Factory Method', la clase Creator proporciona una funcionalidad genérica independientemente del tipo de producto que se quiera crear.
   (a) Verdadero.
   (b) Falso, esa responsabilidad corresponde a la clase ConcreteCreator.
   (c) Falso, esa responsabilidad corresponde a la clase ConcreteProduct.

10. ¿Qué inconveniente tiene el uso del patrón GOF 'Abstract Factory'?
    (a) Los productos de distintas familias comparten el mismo interfaz.
    (b) No es fácil añadir nuevos productos.
    (c) Para el cliente no es fácil seleccionar la familia de productos a crear.

11. ¿Cuándo es necesario introducir una clase 'Fabricación Pura'?
    (a) Cuando queremos una clase intermediaria para desacoplar dos clases ya existentes.
    (b) Cuando el aumento de responsabilidades de una clase pone en peligro su cohesión.
    (c) Cuando necesitamos aplicar el patrón 'Experto en información' en cascada.

12. ¿Qué inconveniente tiene el uso del patrón GOF 'Composite'?
    (a) El cliente no distingue entre clases simples y compuestas.
    (b) No permite añadir nuevas clases simples.
    (c) Resulta complicado imponer restricciones sobre la estructura de los objetos compuestos.

13. En una arquitectura de tipo Microkernel...
    (a) la escalabilidad es alta debido al pequeño tamaño del núcleo.
    (b) cada plugin debe tener un interfaz independiente de los demás.
    (c) se pueden añadir nuevas funcionalidades en tiempo de ejecución.

14. ¿Con qué patrón podemos reducir el acoplamiento entre dos partes de un sistema, cuando una parte usa un conjunto de clases de la otra?
    (a) Composite
    (b) Façade
    (c) Proxy

15. Un lenguaje específico de dominio (DSL)...
    (a) se construye a partir de un diagrama de clases UML.
    (b) puede representar todo o una parte del dominio para el que está construido.
    (c) es menos abstracto que un lenguaje de propósito general.

16. ¿Qué patrón permite centralizar en una sola clase las comprobaciones comunes en una implementación del patrón MVC (seguridad, personalización, etc.)?
    (a) Front Controller
    (b) Page Controller
    (c) Application Controller

17. ¿Qué ventaja NO podemos conseguir con el patrón GOF 'Proxy'?
    (a) Controlar el acceso a un objeto.
    (b) Simular un objeto remoto de forma local.
    (c) Proporcionar una implementación alternativa para un objeto.

18. ¿Quién debe ser el encargado de crear los objetos de transferencia de datos (DTO)?
    (a) El objeto de dominio que contiene los datos.
    (b) Un objeto proxy que ocupa el lugar del DTO.
    (c) Un objeto Assembler que tiene acceso a los objetos de dominio.

19. ¿Qué beneficio obtenemos al usar patrones de software?
    (a) Disminuye la complejidad de los diseños.
    (b) Se reduce la cantidad de código que hay que crear.
    (c) Se simplifica la introducción de nuevas funcionalidades en el futuro.

20. Los objetos de transferencia de datos (DTO)...
    (a) se usan para pasar información entre distintas capas.
    (b) contienen los métodos CRUD que se comunican con la base de datos.
    (c) pueden contener tanto datos como métodos de lógica de negocio.

21. ¿En qué se basa el patrón GRASP 'Polimorfismo'?
    (a) Las instancias de clases hijas se pueden comportar como si se tratase de la clase padre.
    (b) Las instancias de una clase padre se pueden comportar como si se tratase de una clase hija.
    (c) La introducción de una jerarquía de herencia disminuye el acoplamiento.

22. ¿En qué consiste el patrón GRASP 'Indirección'?
    (a) Invertir el flujo de llamadas entre dos clases, facilitando así la automatización de pruebas.
    (b) Separar un conjunto grande de responsabilidades en dos clases distintas, favoreciendo así la cohesión.
    (c) Asignar una responsabilidad a una clase intermedia, desacoplando así dos clases del sistema.

23. ¿En qué se diferencian un modelo de dominio y un diagrama de diseño de clases?
    (a) El modelo de dominio se deriva a partir del diagrama de diseño de clases.
    (b) El diagrama de diseño de clases se deriva a partir del modelo de dominio.
    (c) El modelo de dominio asigna responsabilidades a las clases, mientras que el diagrama de diseño de clases únicamente identifica las relaciones entre clases.

24. En el patrón GOF 'Builder', ¿qué clase debe conocer la secuencia de pasos necesaria para construir un producto?
    (a) La clase Builder.
    (b) La clase ConcreteBuilder.
    (c) La clase Director.

25. ¿Con qué otro patrón GRASP está relacionado el patrón 'Creador'?
    (a) Bajo acoplamiento, ya que disminuye el número de dependencias del sistema.
    (b) Alta cohesión, ya que aumenta la cohesión de la clase creadora.
    (c) Controlador, ya que la clase creadora puede controlar a la clase creada.

26. ¿Qué tipo de acoplamiento debemos evitar para facilitar los cambios de tecnología en la capa de presentación?
    (a) Que la capa de presentación tenga como dependencias clases de la capa de lógica de negocio.
    (b) Que la capa de lógica de negocio tenga como dependencias clases de la capa de presentación.
    (c) Que las clases de la capa de presentación tengan dependencias entre sí.

27. ¿En qué caso está más indicado aplicar el patrón GRASP 'Creador'?
    (a) Cuando queremos limitar el número de instancias de la clase creada.
    (b) Cuando la creación de instancias sigue una lógica condicional.
    (c) Cuando queremos almacenar instancias del objeto creado.

28. Al usar el patrón Transaction Script
    (a) Cada procedimiento creado es independiente del resto de capas del sistema.
    (b) Surgen dos tipos de procedimientos: los que acceden a la base de datos y los que se comunican con el resto de capas del sistema.
    (c) Cada procedimiento representa una acción que el usuario puede ejecutar.

29. ¿Cuál de los siguientes tipos de acoplamiento es más fuerte?
    (a) Cuando hay una jerarquía de herencia.
    (b) Cuando una clase implementa un interfaz.
    (c) Cuando una clase recibe una lista de instancias de otra clase como parámetro en un método.

30. ¿Con qué patrón podemos recibir notificaciones cuando cambia el estado de un objeto?
    (a) Adapter
    (b) Observer
    (c) Command