Sobre la caracterización de los componentes de los sistemas de software abierto: elija la respuesta incorrecta

a. Son sistemas orientados a objetos

b. Son sistemas concurrentes

c. Su funcionalidad puede ser extendida independientemente de otros componentes

d. No pueden definir un estado global visible desde el exterior

e. Deben ser capaces de introducir/salir dinámicamente de los componentes

Diferencias entre los patrones de diseño y los marcos para el desarrollo de software basado en componentes elija la respuesta correcta:

a. Los patrones de diseño son menos abstractos que los frameworks

b. Un framework es un conjunto de clases que pueden ser modificadas para adaptarse a los requerimientos específicos de una aplicación de software

c. Un framework sólo está formado por una colección de componentes de software

d. Un patrón de diseño puede construirse mediante la aplicación de varios frameworks

e. Las clases y los objetos sólo pueden pertenecer a un framework

¿Cuál de las siguientes características no pertenece a un componente de software?

a. Puede componerse con otros componentes en la última fase posible de implementación y despliegue del software

b. Un componente se utiliza para propiciar la reutilización del software mediante acuerdos de nivel de servicio (SLA) o contratos con los clientes

c. Tiene que ser capaz de modificar su propio estado mediante el mecanismo llamado reflexión

d. Es una entidad de software polimórfica que reacciona de forma diferente cuando se despliega en diferentes entornos o contextos de ejecución / Un componente software no puede contener objetos

e.Un componente software no puede contener objetos

f. Puede intercambiarse con otro componente cuya interfaz satisfaga el mismo contrato-cliente o al del primero

Seleccionar la única respuesta incorrecta en relación con los componentes y los objetos:

a. Un marco de trabajo basado en componentes es sólo una caja negra que acepta componentes de software conectados.

b.Un marco de trabajo se puede ver como un diseño reutilizable de un sistema que incluye clases y la forma en que estas se comunican.

c. Un componente puede contener varios elementos del lenguaje de programación: clases, prototipos-objetos y otros recursos asociados (variables, constantes, imágenes, etc.), y no todos estos elementos pertenecen a un lenguaje de programación orientado a objetos.

d. Un componente también puede verse como una arquitectura de software casi completa que contiene algoritmos de uso común dentro de un determinado dominio de aplicación

e. Las superclases de una determinada clase no necesitan pertenecer al mismo componente que la superclase

f. El estado de los objetos y componentes sólo puede obtenerse a través de sus referencias a objetos que contiene.

¿Cuál de las siguientes afirmaciones sobre la PBC (programación basada en componentes) es correcta?

a. La POO define los objetos como entidades de programación pensadas para obtener la reutilización de los campos de la clase (atributos y métodos) únicamente.

b. Los componentes no pueden modificar nunca los recursos que contienen

c. La reutilización de componentes de software significa que un componente no puede ser utilizado más de una vez en la implementación de un sistema/aplicación de software

d. La reutilización de los componentes de software significa que un componente puede ser utilizado más de una vez, pero sin réplicas, dentro de una misma aplicación

e.. Los componentes de software aceptan un tipo de polimorfismo que no puede ser resuelto hasta el momento de la ejecución del programa

f. El "overriding" o la "sobrecarga" de métodos en la POO no son compatibles con el encapsulamiento requerido de los componentes de software.

Seleccionar la única respuesta incorrecta en relación con la especificación de la interfaz de los componentes:

a. Un componente de software puede tener varias interfaces

b. Sólo con la información incluida en la interfaz del componente, los usuarios no pueden conocer el comportamiento del componente dentro de un marco determinado

c. La especificación concreta de las dependencias de los componentes de software sólo dependerá del modelo de componentes que utilice un marco específico

d. Para que un componente sea reutilizable no puede mantener ninguna dependencia de uso con su entorno (contexto de uso)

e. Los futuros estándares de software basados en componentes tendrán como consecuencia un mercado de componentes de software menos robustos y más reutilizables

Seleccionar la respuesta correcta en relación a la especificación de interfaces de los componentes software

(a) Un componente software ha de poseer una sola interfaz

(b) Sólo con la información de la interfaz, los usuarios de un componente no pueden conocer su comportamiento dentro de un marco determinado

(c) Los componentes pueden definir dependencias de uso con otros objetos de la aplicación, sólo han de indicarlo como metainformación de la interfaz

(d) La especificación de las dependencias de un componente puede asumir distintos modelos de componentes dentro de un marco de trabajo determinado

(e) Los diferentes modelos de componentes (.NET, Orbix, JavaBeans...) son incompatibles entre sí

Cuál de los siguientes aspectos no es necesario para obtener la propiedad conocida como interoperabilidad de componentes:

a. Cómo se localizan y prestan los servicios de los componentes

b. Cómo se gestionará la evolución y el mantenimiento de los componentes en el futuro

c. Conocer la forma de especificar correctamente las interfaces entre componentes

d. Que haya compatibilidad binaria en las llamadas a los servicios de los componentes al pasar los parámetros

e. Manejar correctamente las referencias entre objetos remotos cuando el espacio de direcciones del objeto se extiende más allá de los límites de un proceso

Seleccione la única respuesta incorrecta en relación con los modelos de componentes de software:

a. Definen la forma y los mecanismos de interacción entre las interfaces de los componentes

b. Todo marco de componentes debe basarse únicamente en un modelo específico de componentes

c. Los marcos actuales pueden incluir componentes geográficamente remotos (distribución mundial de componentes)

d. La implementación de un sistema de software basado en componentes puede considerarse como la instanciación de un marco a través de un conjunto de componentes ("plugins") y una arquitectura específica que resuelven un problema determinado

e. Cada modelo de componentes (por ejemplo: DCOM, JavaBeans, CORBA) sólo se define dentro de una plataforma específica de distribución de componentes (ActiveX/OLE, Enterprise Beans, Orbix)

Respecto de la propiedad conocida como de extensibilidad independiente de los componentes software, seleccionar la respuesta incorrecta

(a) Dicha propiedad refleja el hecho de que en un sistema de software abierto se pueden introducir nuevos componentes sin que sea necesario realizar posteriormente una comprobación de integridad global de tal sistema

(b) Extensibilidad independiente también significa que los componentes de un software pueden ser desarrollados por empresas independientes de la que suministra el software

(c) Los componentes han de integrarse en un sistema abierto antes de la fase de generación del archivo ".war" con el producto software porque hay que indicar tal inclusión en un archivo de dependencias

(d) La combinación de componentes software ya desarrollados no puede alterar el comportamiento individual de cada uno de los componentes participantes antes de integrarse en tal combinación

(e) Si un componente software no se puede descubrir dinámicamente por un sistema abierto en cualquier instante de ejecución, entonces no es realmente un componente

Respecto de las tecnologías actuales de componentes, elegir la respuesta correcta

(a) En el modelo de componentes de Microsoft, un DLL es una librería del sistema operativo y sólo proporciona su funcionalidad para las operaciones propias de este

(b) Los archivos del sistema: .xdoc, .ocx, .cpl y drivers pueden implementarse con DLLs

(c) Las dependencias entre DLLs pueden evitarse para no violar la auto–contención de estos componentes

(d) Los DLLs sólo se pueden cargar durante la ejecución de una aplicación porque dependen del sistema operativo

(e) Siempre hay que resolver toda la información que necesita un DLL para ejecutarse en el momento de la carga del mismo