3a GUIA DE POO

(Roberto Tecla)

VÁZQUEZ MORENO MARCOS OSWALDO

2CM3

I- Mencione 3 de los elementos esenciales de un **patrón de diseño**.

1.-creacionales

2.-estructurales

3.-comportamiento

II- Mencione los 3 **patrones de diseño** que usa **MVC**-Smalltalk.

1.-modelo

2.-vista

3.-controlador

1.-El operador **new**

|  |  |
| --- | --- |
| a) Asigna memoria a un objeto e inicializa dicho objeto | ( C ) |
| b) Asigna memoria a un objeto |  |
| c) Asigna memoria a un objeto y llama al constructor para que este inicialize a dicho objeto |  |
| d) Ninguno de los anteriores |  |

2.-En Java \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **solo contiene constantes y métodos abstractos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A) Una clase abstracta | B) Una abstracción | C) Una interfaz | D) Una operación | ( C ) |

3.-Cual de las siguientes afirmaciones es falsa para una **clase abstracta**

|  |  |
| --- | --- |
| a) Puede contener variables métodos abstractos y no abstractos | ( A ) |
| b) solo contiene constantes y métodos abstractos |  |
| c) No puede tener instancias directas |  |

4.-En **MVC**-Smalltalk que patrón de diseño **desacopla** las **vistas** de los **modelos**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Compuesto | ( A ) |

5.-En **MVC**-Smalltalk que patrón de diseño **desacopla** las **vistas** de los **controladores**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Compuesto | c) Estrategia | d) Singleton | ( C ) |

6.-En **MVC**-Smalltalk que patrón de diseño permite tratar a una CompositeView como tratamos a uno de sus **componentes**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Compuesto | ( A ) |

7.-En java cuando **un flujo** de E/S **envuelve a otro flujo** de E/S para **agregar funcionalidad** (comportamiento) que **patrón de diseño** se está usando.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Compuesto | ( C ) |

8.-Define una **familia de algoritmos**, encapsula cada uno, y los hace intercambiables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Singleton | ( C ) |

9.-Define una **dependencia uno-a-muchos** entre objetos tal que cuando un objeto cambia todos sus dependientes son notificados y actualizados automáticamente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Singleton | ( A ) |

10.-**Añade responsabilidades adicionales** a un objeto **dinámicamente** y provee una alternativa flexible a la herencia para extender la funcionalidad.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Singleton | ( B ) |

11.**-Garantiza** que una **clase sólo tenga una instancia** y proporciona un punto de acceso global a dicha instancia.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Singleton | ( D ) |

public class Misterio {

private static Misterio var;

private Misterio(){}

public static Misterio getVar(){ if(var==null) { var=new Misterio(); }

return var;

} }

12.-El código de arriba a que **patrón de diseño** corresponde

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Observador | b) Decorador | c) Estrategia | d) Singleton | ( D ) |

**Falso o Verdadero (F/V)**

|  |  |
| --- | --- |
| 1.-La P.O.O. surge para lidiar con la **Complejidad** y el **Cambio** | ( V ) |
| 2.-Todas las **instancias** de una **clase** **comparten la misma estructura** (es decir la clase es una especie de “molde” que le da la misma “forma” a todas las instancias). | ( V ) |
| 3.-Para **invocar** a un **método estático** no se necesita crear un objeto de la clase en la que se define dicho método. | ( F ) |
| 4.-La **herencia** es una relación **IS\_A** | ( V ) |
| 5.-Lo que hace **obsoleto al software** es el **cambio**. | ( V ) |
| 6.-Es un principio de Orientación a Objetos no **encapsular lo que varia** | ( V ) |
| 7.-Es un principio de Orientación a Objetos favorecer **herencia sobre composición**. | ( F ) |
| 8.-Es un principio de O. O. **programar para interfaces no implementaciones**. | ( V ) |
| 9.-El principio **Abierto/Cerrado** establece que: Las entidades de software, como clases, módulos y funciones deben estar **abiertas a modificaciones y cerradas a extensiones**. | ( F ) |
| 10.-El patrón de diseño decorador no satisface el principio Abierto/Cerrado. | ( V ) |
| 11.-El patrón de diseño decorador no depende de la composición. | ( F ) |
| 12.-Una **interfaz**, en Java, solo contiene **constantes y métodos abstractos**. | ( F ) |
| 13.-Una **clase** no puede **implementar más de una interfaz**. | ( V ) |