

Un pacman se mueve de espacio en espacio y tiene un comportamiento diferente según si el espacio tiene un fantasma, comida o su hay pared:

- Si un espacio tiene comida y pasa pacman , deja de tenerla
- Si hay un fantasma, pacman muere
- Si hay una pared no puede ocuparse ese espacio

A. La generalización podría estar directamente en la clase espacio

C. Item.Ocupar() debería devolver un Espacio

E. A y C son correctas



1. ¿Qué es un algoritmo?

a. Un algoritmo es una secuencia de pasos ordenados para resolver un problema

2. ¿Qué es el polimorfismo?

a. Es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación

3. Definición de diagrama de secuencia

4. Cual de las siguientes sentencias representa a una interfaz

a. Define solamente la firma de los métodos

5. El patrón State

a. Es un patrón de comportamiento

b. ME permite eliminar IFs

c. Delega el comportamiento de una clase en su estado

- d. Todas las anteriores
- 6. El patrón State se implementa cuando:
  - a. Se debe agregar o cambiar el comportamiento de un objeto en tiempo de ejecución ante algún cambio de su estado interno
- 7. El patrón Adapter
  - a. Me permite adaptar interfaces incompatibles
- 8. Cual de las siguientes afirmaciones son correctas
  - a. Nuestros diseños deben apuntar a cohesión funcional o perfecta
  - b. El grado de alta cohesión se relaciona directamente con la robustez, la fiabilidad, la reutilización y el grado de compresión
  - c. Nuestro código debe tener alta cohesión
  - d. Todas son correctas
- 9. En un mensaje asíncrono
  - a. El objeto no esperará la respuesta del mensaje antes de continuar
- 10. ¿Cómo se llama la interacción que permite que una clase implemente una interface?
  - a. Interacción de realización
- 11. En el patrón Proxy
  - a. El objeto Proxy tiene que tener como mínimo la misma interfaz que el objeto al que representa
- 12. Que patron utilizaría para una clase que cambia su comportamiento dependiendo de su estado interno
  - a. State
- 13. El patrón Composite
  - a. Se utiliza para crear jerarquizar de objetos
  - b. Es de tipo "Estructural"
  - c. Permite que un objeto composite conozca objetos simples y objetos compuestos
  - d. Todas las anteriores son correctas
- 14. De los siguientes patrones, ¿cuál me permite controlar el acceso a un objeto?
  - a. Proxy
- 15. De los siguientes diagramas, cual NO se agrupa dentro de los diagramas estructurales?
  - a. Diagrama de comunicación
- 16. De los siguientes, ¿cuál NO se considera un Patrón Estructural?
  - a. Strategy
- 17. Es patrón Singleton
  - a. Bla bla bla
  - b. Requiere de un constructor privado para su implementación
  - c. Garantiza que va a existir
  - d. B y C son correctas
- 18. Cuando estoy modelando los atributos de un objeto:
  - a. Debe crear únicamente los atributos necesarios para cumplir con la responsabilidad es ese momento
- 19. El patrón Abstract Factory proporciona una forma de encapsular un grupo de fábricas individuales que tienen un tema común sin especificar sus clases concretas
  - a. Verdadero
- 20. De los siguientes, ¿cuál no se considera un Patrón Creacional?
  - a. Observer
- 21. El patrón Builder
  - a. Sirve para crear objetos complejos de distintas maneras pero con los mismos pasos

22. Considere que tiene algunas imágenes en su tarjeta de memoria y necesita transferirlas a su computadora. Para transferirlos, necesita algún tipo de objeto que sea compatible con los puertos de su computadora para que pueda conectar la tarjeta de memoria a su computadora. Si quisiéramos diagramar esto, que Patrón pudiésemos usar?
- Adapter
23. Cuando clasificó a un método como protegido:
- Se puede acceder desde la clase que lo define y desde cualquier otra que herede de esta clase
24. El patrón Abstract Factory
- Me permite crear familias de objetos compatibles entre sí
  - Puede combinarse con los patrones Singleton y Builder
  - A y B son Correctas
25. ¿Cuál de las siguientes afirmaciones es correcta?
- El Singleton puede ser una solución a la proliferación de objetos
26. Cohesión y Acoplamiento
- Los sistemas deberían tener alta cohesión y bajo acoplamiento para garantizar la calidad de su diseño
  - bla bla
  - La cohesión es el grado de relación que existe entre las tareas de un módulo
  - El acoplamiento es el grado de relación que existe entre dos módulos
  - bla bla
  - A, C y D son correctas
27. El patrón Factory Method
- Me permite cambiar el objeto que se va a crear sin que se entere quien necesita crearlo
28. Una clase Abstracta
- No puede ser instanciada
  - Puede solo utilizarse como superclase o como tipo de un objeto contenido en otra clase
  - Se puede declarar métodos abstractos sin cuerpo
  - Todas las anteriores
29. De los siguientes patrones, ¿cuál me facilita la construcción de un objeto complejo?
- Builder
30. Qué es UML
- Es un lenguaje de modelado de sistemas de software
  - Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema
  - Es un lenguaje ofrece un estándar para describir un plano del sistema
  - Todas las anteriores
31. ¿Qué patrón me permite disponer de varios algoritmos para resolver un problema y elegir cual utiliza en tiempo de ejecución?
- Strategy
32. La herencia es un mecanismo que ocurre cuando existe una interacción llamada?
- Generalización / Especialización
33. ¿Cómo se llama el diagrama que describe las interacciones de un grupo de objetos a través del tiempo?
- Diagrama de secuencia
34. De los siguientes, cuál NO se considera un Patrón de Comportamiento?
- Adapter
35. De los siguientes, cuales se considera un Antipatrón
- Fat class → Se trata de dotar a una clase con demasiados atributos y/o métodos, haciéndola responsable de lógicas de negocio que no les correspondan.

36. Se presenta varios enunciado, indique cual pertenece a cada patrón
- Permite proporcionar un sustituto o placeholder para otro objeto → Proxy
  - Crear objetos complejos a través de uno más simple → Builder
  - Permite la creación de objetos de un subtipo determinado a través de una clase → Factory
37. El patrón Strategy
- Permite intercambiar la implementación de algoritmos en tiempo de ejecución
38. De las siguientes premisas, ¿Cuál nos ayudará detectar si estamos violando el Principio de Responsabilidad Unica?
- El número de métodos públicos
39. De los siguientes patrones, cual me permite trabajar objetos de distintas familias de manera que las familia no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se está usando?
- Abstract Factory
40. Seleccione una:
- Es solamente una manera más complicada de hacer "new"
  - Crea familias de objetos compatibles entre sí.
  - Me permite cambiar el objeto que se va a crear sin que se entere quien necesita crearlo
  - No se puede combinar con el patrón Singleton, ya que eso haría que sólo se pueda crear un objeto
41. Definición de diagrama de estados. Un diagrama de estado
- Muestra los estados en los que puede encontrarse un objeto junto con las transiciones entre los mismos.
  - Modela los cambios de estados de un único objeto en el orden que se ejecutan.
  - Vamos a tener tantos diagramas de estados como objetos tenga nuestro sistema.
42. Definición de cohesión
- grado de relación que tiene el objeto con sus responsabilidades. Solo deben haber método funcionales en una clase, no métodos que no estén siendo usados. Se aumenta la cohesión haciendo que el objeto implemente exclusivamente las responsabilidades que le corresponden. de esta manera será más independiente del resto.
43. Definición de acoplamiento
- El acoplamiento es el nivel de dependencia entre las clases.
  - El objetivo perseguido es que dentro del sistema cada clase sea muy independiente y se comunique con el resto de las clases mediante una interfaz pequeña (con pocos parámetros) y bien definida, con el menor acoplamiento posible. Más parámetros, más nivel de acoplamiento
44. Arquitectura de software
- Organización fundamental de un sistema y sus componentes • Relaciones componentes
  - Ambiente y principios que orientan su diseño y evolución
45. Cual es la generalización
- es una relación entre una clase más general y una más específica o especializada.
  - Dependencia: es una relación de uso entre clases. Los enlaces y las asociaciones son los medios de establecer relaciones entre objetos y clases, respectivamente.
46. Cual es la especialización
- se refiere a que las subclases perfeccionan o especializan a la superclase.
47. Objeto
- Es la representación de un ente del dominio del problema. Tienen un determinado estado interno y responsabilidades. Los objetos responden a los pedidos interactuando con los otros objetos que conoce.
48. Abstracción
- Clasificar los objetos, quitando las acciones y propiedades que no son necesarias.

#### 49. Clases

- a. Representa un molde (template) para objetos de un mismo tipo.

#### 50. POO

- a. Encapsulamiento: ocultamiento del estado interno de un objeto. Permite separar qué hacen los objetos (responsabilidad) de cómo lo hacen (implementación).
- b. Herencia: Jerarquía de clases.
- c. Abstracción: expresa las características esenciales de un objeto, las cuales distinguen a un objeto de los demás.
- d. Polimorfismo: permite al objeto emisor del mensaje despreocuparse de quién es exactamente su colaborador, sólo le interesa que sea responsable de llevar adelante la tarea que le encomienda a través del "mensaje".

#### 51. Para que son los patrones de diseño

- a. Son soluciones en base a la experimentación, creadas para problemas de aparición recurrente en escenarios comunes.

#### 52. Como se llama el proceso por el cual en la programación orientada a objetos se trata de hacer un paralelismo con el mundo real

- a. Proceso de Abstracción

#### 53. Como se denomina al conocimiento de lo que sabe hacer un objeto

- a. Responsabilidad

#### 54. Como se llama el principio, que bien aplicado, transforma al objeto en una caja negra

- a. Encapsulamiento

#### 55. ¿PARA QUÉ SON LOS ESTILOS ARQUITECTÓNICOS?

- a. Definen los patrones posibles de las aplicaciones
- b. Permiten evaluar alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales

---

**Creacionales:** se ocupan de solucionar u ofrecer mejoras a los problemas en la creación o instanciación de un solo objeto en particular

Singleton → Se ocupa de que exista una única instancia de un determinado objeto.

Factory Method → Busca simplificar la creación de objetos de una determinada familia. Definirá una interfaz estándar para la construcción de objetos.

Abstract Factory → Busca centralizar la fabricación de objetos de distintas familias en un solo lugar.

Prototype → Nos permite crear una copia (o clon) más o menos exacto de un objeto en memoria directamente (a partir del prototipo), sin necesidad de tener que interactuar con su clase

Builder → Se ocupa de construir objetos complejos.

**Estructurales:** Se ocupan de resolver problemas relacionados a la estructura interna de los objetos

Adapter → Se dice que adapta un determinado objeto (o en forma más general una interfaz), para que pueda ser usado por otro, dado que de otro modo no podría usarlo.

Composite → Permite, ayuda y simplifica la forma de tratar a objetos simples y compuestos de un mismo modo.

Decorator → Suma o añade funcionalidad a un objeto dinámicamente.

Proxy → Permite proporcionar un sustituto o placeholder para otro objeto. Un proxy controla el acceso al objeto original, lo que le permite realizar algo antes o después de que la solicitud llegue al objeto original.

**Comportamiento:** resuelven problemas respecto la interacción y comunicación entre objetos, así como la forma en la que guardan sus algoritmos que estos pueden implementar.

State → Suele presentarse cuando es necesario que un objeto modifique su comportamiento cuando cambia su estado interno.

Strategy → Permite disponer de varios métodos para resolver un problema y permitir elegir cuál de ellos se usará en tiempo de ejecución.

Observer → Define una dependencia entre un objeto determinado (llamado sujeto u observado) a muchos otros (llamados observadores), de manera que cuando se produce un cambio de estado en el sujeto, los observadores que dependen de él para que él resto se actualicen automáticamente.

Memento → Permite volver a estados anteriores de un objeto y/o sistema.

- **Cohesión y acoplamiento**

- Alta Cohesión y un Bajo acoplamiento (alta relación entre las tareas y una baja dependencia).

- **SOLID**

- Single Responsibility: Una clase debe tener una única razón para cambiar.
- Open Closed: deben estar abiertas para ser extendidas y cerradas para ser modificadas.
- Liskov Substitution: Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.
- Interface Segregation: Muchas interfaces cliente específicas son mejores que una interfaz de propósito general.
- Dependency inversion principle: Se debe depender de abstracciones (es decir, qué quiero representar) y no depender de implementaciones (es decir, cómo lo voy a representar). Ejemplo: Regulador, Termómetro, Horno. Alarma, Sensor, Bocina.

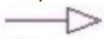







## Acoplamiento

A veces hay acoplamiento entre métodos de la misma clase. Si tengo que conocer algo, ya hay acoplamiento

## Cohesión

Relacionado con la abstracción

Tipo de relaciones:

- Generalización / Especialización
  - Herencia 
  - Interfaz 
- Asociación: No es una relación fuerte. Los tiempos de vida de los objetos son independientes.
  - Unidireccional: Propietario, Inmueble. 
  - Bidireccional: Esposa, Esposo. 
- Agregación: Variante de asociación. Cada uno tiene tiempo de vida fuerte.
  - Carpeta, documento. 
- Composición: Es un tipo de agregación. Cada componente de una agregación puede pertenecer "tan sólo a un todo". Tiempo de vida de la clase contenida, depende de la contenedora.
  - Persona, Órgano. 
- Uso: Una clase utiliza  parámetro para una de sus operaciones. El tiempo de vida de la clase, está ligada a la operación.
  - MP3, Reproductor MP3. 

Arquitectura de software → Organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución

## Arquitectura de capas (presentación - negocio - datos)

- Se definen un conjunto de niveles o capas, en la cual cada nivel interno que se atraviesa se aproxima mas al nivel del conjunto de instrucciones de máquina.
- Sistemas en capas puros: cada capa solo puede comunicarse con la vecina. Aunque esta solución aunque puede ser menos eficiente en algunos casos facilita la portabilidad de los diseños.