Guía de Preguntas – Programación Orientada a Objetos

ALUMNO:
PALUMBO, Mariano

PROFESOR: Dario Cardacci

Curso de Verano Turno NOCHE
Fecha de entrega: 20-03-12

UNIDAD 1

1. Enumere y explique los aspectos más relevantes que hacen que un software de gran magnitud sea complejo.

-La complejidad del dominio del problema: La funcionalidad pura de algunos sistemas es difícil de comprender, y se le añade además todos los requerimientos no funcionales que están implícitos (facilidad de uso, rendimiento, etc). La complejidad externa hace referencia principalmente a "Desacoplamientos de Impedancia" que existen entre los usuarios de un sistema y sus desarrolladores. Los usuarios suelen encontrar grandes dificultades al intentar explicar con precisión sus necesidades en una forma que los desarrolladores puedan comprender. Esto no es culpa del usuario ni del desarrollador, ocurre porque cada uno de los grupos no suele conocer suficientemente el dominio del otro. Ambos tienen perspectivas diferentes sobre la naturaleza del problema y realizan distintas suposiciones sobre la naturaleza de la solución. Una complicación adicional es que los requisitos de un sistema de software cambian frecuentemente durante su desarrollo porque se alteran las reglas del problema.

-La dificultad de gestionar el proceso de desarrollo: nadie puede comprender completamente un sistema a título individual. Incluso si se descompone la implantación de formas significativas, aún hay que enfrentarse a cientos de miles de módulos separados. Esta cantidad de trabajo exige la utilización de un equipo de desarrolladores, y de forma ideal se utiliza un equipo tan pequeño como sea posible. Sin embargo, da igual el tamaño, siempre hay retos considerables asociados con el desarrollo en equipo. Un mayor número de miembros implica una comunicación más compleja y por tanto una coordinación más difícil, particularmente si el equipo está disperso geográficamente. Con un equipo de desarrolladores, el reto clave de la dirección es siempre mantener una unidad e integridad en el diseño.

-La flexibilidad que se puede alcanzar a través del SW: el software ofrece la flexibilidad máxima por lo que un desarrollador puede expresar casi cualquier clase de abstracción. Esta flexibilidad empuja al desarrollador a construir por si mismo prácticamente todos los bloques fundamentales sobre los que se apoyan estas abstracciones de más alto nivel.

-Los problemas que plantea la categorización del comportamiento de sistemas discretos: al ejecutarse el sw en computadores digitales, se tiene un sistema con estados discretos. Los mismos tienen un número finito de estados posibles. Se intenta diseñar los sistemas con una separación de intereses, de forma que el comportamiento de una parte del sistema tenga mínimo impacto en el comportamiento de otra parte del mismo. Todos los eventos externos a un sistema de sw tienen la posibilidad de llevar a ese sistema a un nuevo estado, y más aún, la transición de estado a estado no siempre es determinista. Un evento externo puede corromper el estado del sistema, porque sus diseñadores olvidaron tener en cuenta ciertas interacciones entre eventos. Los eventos externos pueden afectar a cualquier parte del estado interno.

2. ¿Cuáles son los cinco atributos de un sistema complejo?

1. Frecuentemente, la complejidad toma la forma de una jerarquía, por lo cual un sistema complejo se compone de subsistemas, que a su vez tienen relacionados otros subsistemas y así sucesivamente, hasta que se alcanza algún nivel ínfimo de componentes elementales.

2. La elección de qué componentes de un sistema son primitivos es relativamente arbitraria y queda en gran medida a decisión del observador. Lo que es primitivo para un observador puede estar a un nivel de abstracción mucho más alto para otro.

3. Los enlaces internos de los componentes suelen ser más fuertes que los enlaces entre componentes. Este hecho tiene el efecto de separar la dinámica de la estructura interna de los componentes de la dinámica que involucra la interacción entre los componentes. Esta diferencia entre interacciones intracomponentes e intercomponentes proporciona una separación clara de intereses entre las diferentes partes de un sistema, posibilitando el estudio de cada parte de forma relativamente aislada.

4. Los sistemas jerárquicos están compuestos usualmente de solo unas pocas clases diferentes de subsistemas en varias combinaciones y disposiciones. Básicamente se trata de elementos primitivos que se relacionan con elementos más complejos, combinando muchos elementos primitivos formo un elemento complejo. Esto se usa mucho para

reutilización de código. Los sistemas complejos tienen patrones comunes. Estos patrones pueden conllevar la reutilización de componentes pequeños, tales como las células que se encuentran en plantas y animales, o de estructuras mayores como los sistemas vasculares que también aparecen en ambas clases de seres vivos.

5. Se encontrará invariablemente que un sistema complejo que funciona ha evolucionado de un sistema simple que funcionaba. Un sistema complejo diseñado desde cero nunca funciona y no puede parchearse para conseguir que lo haga. Hay que volver a empezar, partiendo de un sistema simple que funcione. A medida que los sistemas evolucionan, objetos que en su momento se consideraron complejos se convierten en objetos primitivos sobre los que se construyen sistemas más complejos aún. Es más, nunca se pueden crear estos objetos primitivos de forma correcta la primera vez: hay que utilizarlos antes en un contexto, y mejorarlos entonces con el tiempo cuando se aprende más sobre el comportamiento real del sistema.

3. ¿Cuáles son las dos jerarquías más importantes que consideramos en la orientación a objetos para sistemas complejos?

•"parte – de" (part of) Estructura de Clases

•"es – un" (is a) Estructura de Objetos

De forma conjunta, nos referimos a la estructura de clases y de objetos de un sistema como su arquitectura.

La Jerarquía es una propiedad que permite la ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son: estructura de clases (jerarquía "es-un" (is-a): generalización/especialización) y estructura de objetos (jerarquía "parte-de" (part-of): agregación).

Las jerarquías de generalización/especialización se conocen como herencia. Básicamente, la herencia define una relación entre clases, en donde una clase comparte la estructura o comportamiento definido en una o más clases (herencia simple y herencia múltiple, respectivamente).

La agregación es el concepto que permite el agrupamiento físico de estructuras relacionadas lógicamente. Así, un camión se compone de ruedas, motor, sistema de transmisión y chasis; en consecuencia, camión es una agregación, y ruedas, motor, transmisión y chasis son agregados de camión

4. ¿Con qué podemos enfrentar a la complejidad para obtener partes cada vez más pequeñas y simplificadas del dominio del problema?

La técnica de dominar la complejidad se conoce como DIVIDE y VENCERAS. Cuando se diseña un sistema de software complejo, es esencial descomponerlo en partes más y más pequeñas, cada una de las cuales se puede refinar entonces de forma independiente. Para entender un nivel dado de un sistema, basta con comprender unas pocas partes (no necesariamente todas) a la vez.

5. ¿Cuáles son las dos formas de descomposición más conocidas?

Las dos formas de descomposición son la Algorítmica y la Orientada a objetos.

6. ¿Explique en qué se diferencia la descomposición algorítmica y la orientada a objetos?

Descomposición algorítimica: = Análisis y diseño estructurado. Se aplica para descomponer un gran problema en pequeños problemas, la unidad fundamental de este tipo de descomposición es el subprograma. El programa resultante toma la forma de un árbol, en el que cada subprograma realiza su trabajo, llamado ocasionalmente a otro subprograma.

Descomposición orientada a objetos= Análisis y diseño Orientado a Objetos. El mundo es visto como un conjunto de entidades autónomas, que al colaborar muestran cierta conducta. Los algoritmos no existen de manera independiente, estos están asociados a los objetos. Cada objeto exhibe un comportamiento propio bien definido, y además modela alguna entidad del mundo real. Los objetos hacen cosas, y se les pide que hagan lo que hacen

enviándoles mensajes. Puesto que esta descomposición está basada en objetos y no en algoritmos, se la llama descomposición orientada a objetos.

Ambas visiones son muy importantes: la visión algorítmica enfatiza el orden de los eventos, y la visión orientada a objetos resalta los agentes que o bien causan acciones o bien son sujetos de estas acciones. Sin embargo, el hecho es que no se puede construir un sistema complejo de las dos formas a la vez, porque son vistas completamente perpendiculares. Hay que comenzar a descomponer un sistema sea por algoritmos o por objetos, y entonces utilizar la estructura resultante como marco de referencia para expresar la otra perspectiva.

7. ¿Qué rol cumple la abstracción en la orientación a objetos?

Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador. Podemos definir la abstracción como una operación intelectual que separa las cualidades de un objeto para considerarlas aisladamente o para analizar al objeto en su pura esencia o noción. Básicamente es la capacidad de separar los elementos (al menos mentalmente) para poder verlos de forma singular. Como cuando describimos el cuerpo humano y decimos cabeza, brazo(s), pierna(s), etc. Gracias a la abstracción podemos representar las características esenciales de un objeto sin preocuparnos de las restantes características (no esenciales). En la abstracción siempre hay información que se pierde. Todo depende de los aspectos que me interesen representar. Una abstracción es una representación. Básicamente es tarea del analista, el programador toma esa abstracción y la traduce a código.

8. ¿Qué rol cumple la jerarquía en la orientación a objetos?

Resalta la estructura y comportamiento comunes en el interior de un sistema. Así por ejemplo, en vez de estudiar cada una de las células de una hoja de una planta específica, es suficiente estudiar 1 de esas células, porque se espera que todas las demás se comporten de modo similar. Aunque se pueda tratar como distinta casa instancia de un determinado tipo de objeto, puede asumirse que comparte la misma conducta que todas las demás instancias de ese mismo tipo de objeto. Clasificando objetos en grupos de abstracciones relacionadas se llega a una distinción explicita de las propiedades comunes y distintas entre diferentes objetos.

9. ¿Consideraría Ud. al diseño orientado a objetos un desarrollo evolutivo o revolucionario? Justifique.

Es un desarrollo evolutivo, no revolucionario; no rompe con los avances del pasado, sino que se basa en avances ya probados.

10. ¿Cuántos y cuáles son los modelos básicos que se manejan en el diseño orientado a objetos?

Son 4 los modelos básicos:

-Modelo estático.

-Modelo dinámico.

-Modelo lógico (estructura de clases y objetos)

-Modelo físico (arquitectura de módulos y arquitectura de procesos)

Los modelos del diseño orientado a objetos reflejan la importancia de plasmar explícitamente las jerarquías de clases y de objetos del sistema que se diseña. Estos modelos cubren también el espectro de las decisiones de diseño relevantes que hay que considerar en el desarrollo de un sistema complejo, y así animan a construir implantaciones que poseen los cinco atributos de los sistemas complejos bien formados.

11. ¿Qué es la programación orientada a objetos?

Es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia.

MUY IMPORTANTE: La POO utiliza objetos no algoritmos, como sus bloques lógicos de construcción fundamentales (la jerarquía es parte de). Cada objeto es una instancia de alguna clase; y las clases están relacionadas con otras clases por medio de relaciones de herencia (jerarquía de clases). Si faltan cualquiera de estos 3 elementos un programa NO es orientado a objetos.

12. ¿Qué es el diseño orientado a objetos?

Es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estáticos y dinámicos del sistema que se diseña.

MUY IMPORTANTE: el diseño orientado a objetos da lugar a una descomposición orientada a objetos y utiliza diversas notaciones para expresar diferentes modelos del diseño lógico (estructura de clases y objetos) y físicos (arquitectura de módulos y procesos) de un sistema, además de los aspectos estáticos y dinámicos del sistema. El soporte para la descomposición orientada a objetos es lo que hace el diseño orientado a objetos bastante diferente del estructurado: el primero utiliza abstracciones de clases y objetos para estructurar lógicamente los sistemas, y el segundo utiliza abstracciones algorítmicas.

13. ¿Qué es el análisis orientado a objetos?

Es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema.

AOO, DOO y POO se relacionan de la siguiente manera: básicamente los productos del AOO sirven como modelos de los que se puede partir para un diseño orientado a objetos; los productos del diseño orientado a objetos pueden utilizarse entonces como anteproyectos para la implementación completa de un sistema utilizando métodos de programación orientada a objetos.

14. ¿Cuáles son los elementos fundamentales en el modelo de objetos?

1. Abstracción
2. Encapsulamiento
3. Modularidad
4. Jerarquía

Fundamentales significa que un modelo que carezca de cualquiera de estos elementos no es orientado a objetos.

15. ¿Cuáles son los elementos secundarios del modelo de objetos?

1. Tipos
2. Concurrencia
3. Persistencia

Por secundarios quiere decirse que cada uno de ellos es una parte útil del modelo de objetos, pero no es esencial.

16. Explique el significado de la abstracción.

Es una de las vías fundamentales por la que los humanos combatimos la complejidad. La abstracción se centra en las características esenciales de algún objeto en relación a la perspectiva del observador.

“Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objeto y proporciona así fronteras nítidamente definidas respecto a la perspectiva del observador”.

Una abstracción se centra en la visión externa de un objeto, y, por tanto, sirve para separar el comportamiento esencial de un objeto de su implantación.

La abstracción consiste en captar las características esenciales de un objeto, así como su comportamiento. Por ejemplo, ¿Qué características podemos abstraer de los automóviles? O lo que es lo mismo ¿Qué características semejantes tienen todos los automóviles? Todos tendrán una marca, un modelo, número de chasis, peso, llantas, puertas, ventanas, etc. Y en cuanto a su comportamiento todos los automóviles podrán acelerar, frenar, retroceder, etc. En los lenguajes de programación orientada a objetos, el concepto de Clase es la representación y el mecanismo por el cual se gestionan las abstracciones. La abstracción es uno de los medios más importantes mediante el cual nos enfrentamos con la complejidad inherente al software. La abstracción es la propiedad que permite representar las características esenciales de un objeto sin preocuparse de las restantes características (no esenciales). La abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación.

En la abstracción siempre hay información que se pierde, todo depende de los aspectos que me interesen representar. Una abstracción es una representación. Básicamente es tarea del analista, el programador toma esa abstracción y la traduce a código.

17. Explique el significado del encapsulamiento.

La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se centra en el comportamiento observable de un objeto, mientras el encapsulamiento se centra en la implementación que da lugar a este comportamiento. El encapsulamiento se consigue mediante la ocultación de información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; típicamente, la estructura de un objeto está oculta, así como la implantación de sus métodos.

Cada clase debe tener dos partes: una interfaz y una implementación. La interfaz de una clase captura sólo su vista externa, abarcando la abstracción que se ha hecho del comportamiento común de todas las instancias de la clase. La implementación (implantación) de una clase comprende la representación de la abstracción así como los mecanismos que consiguen el comportamiento deseado. La interfaz de una clase es el único lugar en el que se declaran todas las suposiciones que un cliente puede hacer acerca de todas las instancias de la clase; la implantación encapsula detalles acerca de los cuales ningún cliente puede realizar suposiciones.

“El encapsulamiento es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar la interfaz contractual de una abstracción y su implantación”.

El encapsulamiento consiste en unir en la Clase las características y comportamientos, esto es, las variables y métodos. Es tener todo esto es una sola entidad. En los lenguajes estructurados esto era imposible. La utilidad del encapsulamiento va por la facilidad para manejar la complejidad, ya que tendremos a las Clases como cajas negras donde sólo se conoce el comportamiento pero no los detalles internos, y esto es conveniente porque nos interesará será conocer qué hace la Clase pero no será necesario saber cómo lo hace.

18. Explique el significado de la modularidad.

“La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados”.
Básicamente se trata de hacer lo posible por construir módulos cohesivos (agrupando abstracciones que guarden cierta relación lógica) y débilmente acoplados (minimizando las dependencias entre los módulos). Así, los principios de abstracción, encapsulamiento y modularidad son sinérgicos. Un objeto proporciona una frontera bien definida alrededor de una sola abstracción, y tanto el encapsulamiento como la modularidad proporcionan barreras que rodean a esta abstracción.
El concepto de modularidad, se refiere a una organización en la que distintos componentes de un sistema de programación se dividen en unidades funcionales separadas.
La modularidad tiene cohesión (relaciones fuertes entre elementos. Tiene mucha especificad, la característica principal es que es específico. Cuando algo es poco cohesivo no es muy reutilizable. Lo contrario a cohesión es

acoplamiento) y débilmente acoplado (un módulo que no depende de otro módulo es muy reutilizable.). Muy importante: *Lograr fuertes relaciones internas y muy pocas dependencias o relaciones externas.*

19. Explique el significado de la jerarquía.

"La jerarquía es una clasificación u ordenación de abstracciones"
Las dos jerarquías más importantes en un sistema complejo son su estructura de clases (la jerarquía de clases) y su estructura de objetos (la jerarquía de partes).
La herencia es la jerarquía <<de clases>> más importante, es un elemento esencial de los sistemas orientados a objetos. Básicamente la herencia define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida en una o más clases (lo que se denomina herencia simple o herencia múltiple, respectivamente). La herencia representa así una jerarquía de abstracciones, en la que una subclase hereda de una o más superclases. Típicamente, una subclase aumenta o redefine la estructura y el comportamiento de sus superclases. La herencia implica una jerarquía de generalización/especialización, en la que una subclase especializa el comportamiento o estructura, más general, de sus superclases. A medida que se desarrolla la jerarquía de herencias, la estructura y comportamiento comunes a diferentes clases tenderá a migrar hacia superclases comunes. Por esta razón se habla a menudo de la herencia como una jerarquía de generalización/especialización. Las superclases representan abstracciones generalizadas, y las subclases representan especializaciones en las que los campos y métodos de la superclases sufren añadidos, modificaciones o incluso ocultaciones.
Las jerarquías <<parte de>> describen relaciones de agregación. La agregación permite el agrupamiento físico de estructuras relacionadas lógicamente, y la herencia permite que estos grupos de aparición frecuente se reutilicen con facilidad en diferentes abstracciones. La agregación plantea el problema de la propiedad.

20. Explique el significado de la tipificación.

"Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas"
Permite la agrupación de objetos en tipos. Un tipo es una caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades. La idea de congruencia es central a la noción de tipos (ejemplo de unidades de medidas en física). La comprobación estricta de tipos impide que se mezclen abstracciones. Los tipos permiten expresar las abstracciones de manera que el lenguaje de programación en el que se implantan puede utilizarse para apoyar las decisiones de diseño.
Tipos != de Clase (son distintos, no es lo mismo!!!)
Objetos de diferentes tipos no se puedan intercambiar, o se pueden intercambiar solo de forma restringida. Tipo es una caracterización precisa de las propiedades estructurales y de comportamiento que comparten una colección de entidades. Existen lenguajes fuertemente tipificados (Ada) y débilmente tipificados. Estos últimos soportan polimorfismo, mientras que los fuertemente tipificados no.

21. Explique el significado de la concurrencia.

"La concurrencia es la propiedad que distingue un objeto activo de uno que no está activo"
Mientras la POO se centra en la abstracción de datos, encapsulamiento y herencia, la concurrencia se centra en la abstracción de procesos y la sincronización. El objeto es un concepto que unifica estos dos puntos de vista distintos: cada objeto (extraído de una abstracción del mundo real) puede representar un hilo separado de control (una abstracción de un proceso). Tales objetos se llaman activos. En un sistema basado en diseño orientado a objetos, se puede conceptualizar el mundo como un conjunto de objetos cooperativos, algunos de los cuales son activos y sirven así como centros de actividad independiente.
Concurrencia permite que diferentes objetos actúen al mismo tiempo, usando distintos threads de control.

22. Explique el significado de la persistencia.

"La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado)" .El tiempo de vida del objeto trasciende al tiempo de vida del programa que los creó.
La persistencia conserva el estado de un objeto en el tiempo y en el espacio.

La persistencia abarca algo más que la mera duración de los datos. En las BD orientadas a objetos, no sólo persiste el estado de un objeto, sino que su clase debe trascender también a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado almacenado.

23. ¿Cómo se denotan las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador?

Abstracción

24. ¿A qué denominamos un objeto cliente?

Un objeto puede operar sobre otros objetos pero otros objetos no pueden operar sobre él. (Operar=mandar mensajes)

25. ¿A qué denominamos un objeto servidor?

Un objeto que nunca opera sobre otros objetos, sólo otros objetos operan sobre él. No puede invocar métodos de otros, solo recibe invocaciones.

26. ¿A qué denomina Meyer el modelo contractual de programación?

La vista exterior de cada objeto define un contrato del que pueden depender otros objetos, y a su vez debe ser llevado a cabo por la vista interior del propio objeto (a menudo con la colaboración de otros objetos). Así, este contrato establece todas las suposiciones que puede hacer un objeto cliente acerca del comportamientos de un objeto servidor. Al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con las formas de invocación u órdenes que admite, se le denomina su protocolo. Un protocolo denota las formas en las que un objeto puede actuar y reaccionar y de esta forma constituye la visión externa completa, estática y dinámica, de la abstracción.

27. ¿Qué establece un contrato entre objetos?

Establece las responsabilidades de un objeto.

28. ¿Cómo se denomina a las formas en que un objeto puede actuar y/o reaccionar, constituyendo estas formas la visión externa completa, estática y dinámica de la abstracción?

Protocolo.

29. ¿Cómo se denomina al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con las formas de invocación u órdenes que admite?

Protocolo. Al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con las formas de invocación u órdenes que admite, se le denomina protocolo. Un protocolo denota las formas en las que un objeto puede actuar y reaccionar y de esta forma constituye la visión externa completa, estática y dinámica, de la abstracción.

30. ¿A qué nos referimos cuando decimos que un concepto central de la idea de abstracción es el de invariancia?

Un invariante es una condición booleana cuyo valor de verdad debe mantenerse. Si un cliente viola su parte del convenio el servidor no puede proceder con fiabilidad. Del mismo modo, si el servidor no lleva a cabo su parte del contrato, sus clientes ya no pueden seguir confiando en el comportamiento del servidor. Entonces ese valor de verdad debe mantenerse y si se rompe esa invariancia, se rompe el contrato asociado con una abstracción.

31. ¿Qué se debe definir para cualquier operación asociada a un objeto?

Se deben definir pre y post condiciones.

32. ¿Qué es una precondición?

La precondición es una condición que debe cumplirse cuando la operación inicia su ejecución. En caso de no cumplirse, entonces la operación podrá de manera legítima rehusar su ejecución y posiblemente disparar alguna excepción.

33. ¿Qué es una postcondición?

La postcondición es una condición que debe ser verdadera cuando la operación termina su ejecución. Sí no es verdadera, entonces la implementación de la operación tiene defectos y deberán ser corregidos.

Bertrand Meyer y otros autores describen las pre- y postcondiciones de una operación como un contrato entre una operación y un cliente que envía un mensaje a esa operación. La metáfora del contrato implica que:

1- Si el transmisor (sender) del mensaje puede garantizar que la precondición sea verdadera, entonces el receptor (target) encargado de la operación garantizará que la postcondición sea verdadera después de la ejecución.

2- Si, por otro lado, el transmisor del mensaje no puede garantizar que la precondición sea verdadera, entonces el contrato completo se ha violado y es inválido: La operación no está obligada a ser ejecutada y de serlo a garantizar la postcondición.

Debemos puntualizar que un invariante de clase es verdadero tanto cuando una operación inicia su ejecución y cuando la termina.

Los invariantes de clase, junto con las pre- y postcondiciones de las operaciones forman un marco de trabajo (framework) para el proceso de diseño conocido como Diseño por Contrato, el cual garantiza que la operación del objeto receptor generará una respuesta correcta a un mensaje enviado a él y que el cliente ha obedecido las precondiciones de la operación.

34. ¿A qué se denomina excepción?

Es una indicación de que no se ha satisfecho algún invariante. Una excepción se genera cuando se produce un acontecimiento circunstancial que impide el normal funcionamiento del programa, es un comportamiento no deseado.

35. ¿A qué se denomina mensaje?

Un mensaje es simplemente una operación que un objeto realiza sobre otro. El mensaje es esencialmente una orden que se envía a un objeto para indicarle que realice alguna acción. Esta técnica de enviar mensajes a objetos se denomina paso de mensajes. Un mensaje es una petición de un objeto a otro objeto al que le solicita ejecutar uno de sus métodos. Por convenio, el objeto que envía la petición se denomina emisor y el objeto que recibe la petición se denomina receptor.

36. ¿El encapsulado es un concepto complementario a la abstracción? Justifique.

Si. La abstracción se centra en el comportamiento observable de un objeto, mientras en encapsulamiento se centra en la implementación que da lugar a este comportamiento. El encapsulamiento se consigue a menudo mediante la ocultación de información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; típicamente, la estructura de un objeto está oculta, así como la implantación de sus métodos.

37. ¿Cómo se denomina al elemento de un objeto que captura su vista externa?

Interfaz

38. ¿Cómo se denomina al elemento de un objeto que captura su vista interna la cual incluye los mecanismos que consiguen el comportamiento deseado?

Implementación

39. ¿El concepto de "ocultar los detalles de implementación" lo asociaría a "esconder los detalles de implementación" o a "evitar el uso inapropiado de los detalles de implementación"? Justifique.

Se asocia a evitar el uso inapropiado de los detalles de implementación. Se trata de tener encapsulada esa información en el estado interno del objeto. Nos interesa saber qué hace pero no cómo lo hace.

40. ¿Cuáles son los dos aspectos que hacen importante considerar a la modularidad?

La cohesión y que los módulos estén débilmente acoplados.

41. ¿Para qué se utiliza la jerarquía?

Para ordenar abstracciones.

42. ¿Cómo denominamos a la caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades?

Tipo

43. ¿Las clases implementan tipos?

Sí

44. ¿Los tipos implementan clases?

No

45. ¿Cómo denominamos a los lenguajes que hacen una comprobación de tipos estricta?

Lenguajes Tipados

46. ¿Cómo denominamos a los lenguajes que no hacen una comprobación de tipos estricta?

Débilmente tipados

47. ¿A qué se denomina ligadura estática (temprana)?

Se refiere al momento en el que los nombres se ligan con sus tipos. La ligadura estática significa que se fijan los tipos de todas las variables y expresiones en tiempo de compilación. El compilador garantizará que el programa se ejecutará sin errores de tipos.

48. ¿A qué se denomina ligadura dinámica (tardía)?

Significa que los tipos de las variables y expresiones no se conocen hasta el tiempo de ejecución.

49. ¿Es lo mismo la comprobación estricta de tipos y la ligadura dinámica?

No. Son conceptos independientes. Un lenguaje puede tener comprobación estricta de tipos y tipos estáticos, puede tener comprobación estricta de tipos pero soportar enlace dinámico, o no tener tipos y admitir la ligadura dinámica.

La comprobación de tipos comprueba que el tipo de una construcción tenga sentido en su contexto según el lenguaje

50. ¿Cómo se denomina la característica que permite a diferentes objetos actuar al mismo tiempo?

Concurrencia

51. ¿A qué se denomina concurrencia pesada?

Un proceso pesado es aquel típicamente manejado de forma independiente por el sistema operativo de destino, y abarca su propio espacio de direcciones. Es un programa más, un proceso más.

52. ¿A qué se denomina concurrencia ligera o liviana?

Un proceso ligero suele existir dentro de un solo proceso del SO en compañía de otros procesos ligeros, que comparten el mismo espacio de direcciones. La comunicación entre procesos pesados suele ser costosa, involucrando a alguna forma de comunicación interproceso; la comunicación entre procesos ligeros es menos costosa, y suele involucrar datos compartidos.

53. ¿La concurrencia es la propiedad que distingue un objeto activo de uno que no lo está?

Sí.

54. ¿Cómo se denomina la característica en orientación a objetos que permite conservar el estado de un objeto en el tiempo y el espacio?

Persistencia.

55. ¿Qué cosas se persisten?

En las BD orientadas a objetos, no sólo se persiste el estado de un objeto, sino que su clase debe trascender también a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado almacenado.

56. Defina qué es un objeto desde la perspectiva de la cognición humana.

*Una cosa tangible y/o visible
*Algo que puede comprenderse intelectualmente.
* Algo hacia lo que se dirige un pensamiento o acción.

Un objeto modela alguna parte de la realidad y es, por tanto, algo que existe en el tiempo y el espacio.

57. ¿Un objeto es real o abstracto? Justifique.

Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema. Es cualquier cosa real o abstracta que posee una estructura que lo define y acciones que lo controlan.

58. ¿Los objetos poseen límites físicos precisos o imprecisos?

Puede poseer las dos cosas.

59. ¿Cuáles son las tres cosas que debe tener un objeto?

Un objeto tiene estado, exhibe algún comportamiento bien definido, tiene una identidad única.

60. ¿Cuál es la palabra que se puede utilizar como sinónimo de objeto?

Instancia

61. ¿Cuál es la palabra que se puede utilizar como sinónimo de instancia?

Objeto

62. ¿Cómo definiría el estado de un objeto?

El estado de un objeto abarca todas las propiedades (normalmente estáticas) del mismo más los valores actuales (dinámicos) de cada una de esas propiedades. Básicamente es el conjunto de todas las propiedades estáticas y los valores dinámicos que adoptan en un momento dado. El hecho de que todo objeto tiene un estado implica que todo objeto toma cierta cantidad del espacio, ya sea en el mundo físico o en la memoria del computador.

63. ¿A qué definimos propiedad de un objeto?

Una propiedad es una característica distintiva, un rasgo o cualidad que contribuye a hacer que un objeto sea ese objeto y no otro. Por ejemplo, una propiedad esencial de un ascensor es que está restringido a moverse arriba y abajo y no horizontalmente.

64. ¿Qué es lo que distingue a “un objeto” de los “valores simples”?

Todas las propiedades tienen algún valor, este valor puede ser una mera cantidad, o puede denotar a otro objeto. Así, se distingue entre objetos y valores simples: las cantidades simples como el número 3 son intemporales, inmutables y no instanciadas, mientras que los objetos existen en el tiempo, son modificables, tienen estado, son instanciados y pueden crearse, destruirse y compartirse.

65. ¿Cómo definiría el comportamiento de un objeto?

Ningún objeto existe de forma asilada. En vez de eso, los objetos reciben acciones, y ellos mismos actúan sobre otros objetos. “El comportamiento es cómo actúa y reacciona un objeto, en términos de sus cambios de estado y paso de mensajes”

El comportamiento de un objeto representa su actividad visible y comprobable exteriormente. Una operación es una acción que un objeto efectúa sobre otro con el fin de provocar una reacción.

66. ¿El comportamiento de un objeto se ve afectado por el estado del mismo o bien que el comportamiento del objeto es función de su estado?

El comportamiento de un objeto es función de su estado así como de la operación que se realiza sobre él, teniendo algunas operaciones efecto lateral de modificar el estado del objeto, entonces el estado de un objeto representa los resultados acumulados de su comportamiento.

67. ¿Algunos comportamientos pueden alterar el estado de un objeto?

Sí.

68. Se puede afirmar que el estado de un objeto termina siendo los resultados acumulados de su comportamiento.

Sí.

69. ¿A qué definiría como operación (método/función miembro)?

Una operación denota un servicio que una clase ofrece a sus clientes. Algún trabajo que un objeto realiza sobre otro con el fin de provocar una reacción. Es una funcionalidad que la clase expone para que otras la puedan usar.

70. ¿Cuáles son las tres operaciones más comunes?

*Modificador
*Selector
*Iterador.

71. ¿Cuáles son las dos operaciones habituales que se utilizan para crear y destruir instancias de clases?

*Constructor
*Destructor

72. ¿Qué tipo de operación es el modificador?

Una operación que altera el estado de un objeto

73. ¿Qué tipo de operación es el selector?

Una operación que accede al estado de un objeto, pero no altera ese estado.

74. ¿Qué tipo de operación es el iterador?

Una operación que permite acceder a todas las partes de un objeto en algún orden perfectamente establecido.

75. ¿Qué tipo de operación es el constructor?

Una operación que crea un objeto y/o inicializa su estado. Es el lugar para inicializar el estado de un objeto. Se usa para crear la contención física.

76. ¿Qué tipo de operación es el destructor?

Una operación que libera el estado de un objeto y/o destruye el propio objeto.

77. ¿Cómo denominamos operaciones fuera de las clases en aquellos programas orientados a objetos que permiten colocarlas (ej. C++)?

C++ permite escribir operaciones como subprogramas libres, llamados funciones no miembro. Los subprogramas libres son procedimientos o funciones que sirven como operaciones no primitivas sobre un objeto u objetos de la misma o de distintas clases.

78. ¿Todos los métodos son operaciones?

Sí, todos los métodos son operaciones.

79. ¿Todas las operaciones son métodos?

No. No todas las operaciones son métodos: algunas operaciones pueden expresarse como subprogramas libres.

80. Dado el protocolo de un objeto (todos sus métodos y subprogramas libres asociados al objeto si el lenguaje lo permite) es conveniente dividirlo en grupos lógicos más pequeños de comportamiento? ¿Por qué?

Todos los métodos y subprogramas libres asociados con un objeto concreto forman su protocolo. El protocolo define así la envoltura del comportamiento admisible en un objeto, y por tanto engloba la visión estática y dinámica completa del mismo. Para la mayoría de las abstracciones no triviales, es útil dividir este protocolo mayor en grupos lógicos de comportamiento. Estas colecciones, que constituyen una partición del espacio de comportamiento de un objeto, denotan los papeles que un objeto puede desempeñar y se definen los contratos entre las abstracciones y sus clientes.

81. ¿Cómo denominamos a los grupos lógicos más pequeños de comportamiento del protocolo total de un objeto?

Papeles o roles.

82. ¿Cuáles son las dos responsabilidades más importantes que posee un objeto?

El conocimiento que un objeto mantiene y las acciones que puede llevar a cabo. Las responsabilidades están encaminadas a transmitir un sentido del propósito de un objeto y de su lugar en el sistema. Las responsabilidades de un objeto son todos los servicios que proporciona para todos los contratos que soporta. Entonces, el estado y comportamiento de un objeto definen en conjunto los papeles que puede representar un objeto en el mundo, los cuales a su vez cumplen las responsabilidades de la abstracción.

83. ¿Es relevante el orden en que se invocan las operaciones de un objeto?

Si por que puede ocurrir, por ejemplo que la máquina expendedora busque indefinidamente una lata de la cual ya no hay stock, antes de buscarla se tiene que fijar si hay o no stock.

84. ¿Por qué decimos que los objetos se pueden considerar como máquinas?

Porque para algunos objetos la ordenación de las operaciones respecto a los eventos y al tiempo es tan penetrante que se puede caracterizar mejor formalmente el comportamiento de tales objetos en términos de una máquina de estados finitos equivalente.

Las instancias de clases, actúan como pequeñas máquinas y, ya que todas esas instancias incorporan el mismo comportamiento, se puede utilizar la clase para capturar esta semántica común de orden respecto al tiempo y los eventos.

85. ¿Qué es la identidad de un objeto?

Es aquella propiedad o conjunto de propiedades que lo distingue de otros objetos.

86. Dadas dos variable X e Y del mismo tipo ¿qué significa que ambas son iguales?

Dos variables que apunten al mismo objeto. Dos variables que apunten a 2 instancias pero con el mismo estado.

87. Dadas dos variable X e Y del mismo tipo ¿qué significa asignarle Y a X?

Y apunta a 1 instancia en memoria o nada y X apunta al mismo lugar**.**

88. Dadas dos variable X e Y del mismo tipo ¿qué significa clonar X en Y?

(Pueden tener el mismo estado o no) Ambas van a apuntar al mismo objeto en memoria.

89. ¿Qué significa realizar una clonación superficial?

Significa que copia el objeto pero comparte su estado.

90. ¿Qué significa realizar una clonación profunda?

Copia el objeto así como su estado y así recursivamente

91. ¿Qué es el ciclo de vida de un objeto?

El ciclo de vida de un objeto se extiende desde el momento en que se crea por primera vez (y consume así espacio por primera vez) hasta que ese espacio se recupera. Para crear explícitamente un objeto, hay que declararlo o bien asignarle memoria dinámicamente.

92. ¿Cómo se libera el espacio ocupado por un objeto?

En lenguajes como VB.NET, un objeto se destruye automáticamente como parte de la recolección de basura cuando todas las referencias a él se han perdido. En lenguajes sin recolección de basura, un objeto sigue existiendo y consume espacio incluso si todas las referencias a él se han perdido. Los objetos creados con New() deben ser destruidos de manera explícita con el operador delete, (se llama al destructor).

93. ¿Qué tipos de relaciones existen entre los objetos?

*Enlace: "conexión física o conceptual entre objetos." Un objeto colabora con otros objetos a través de sus enlaces con éstos. Un enlace denota la asociación específica por la cual un objeto (cliente) utiliza los servicios de otro objeto (servidor), o a través del cual un objeto puede comunicarse con otro. El enlace es una relación semántica, es una relación de uso entre dos elementos, no hay jerarquía, es una relación de iguales.

*Agregación: mientras que los enlaces denotan relaciones igual-a-igual o cliente/servidor, la agregación denota una jerarquía todo/parte, con la capacidad de ir desde el todo (también llamado el agregado) hasta sus partes (atributos). La agregación es un tipo especializado de asociación. "es parte de" (cuando un objeto está comprendido o contenido dentro de otro).

94. ¿Cómo podemos definir al enlace entre objetos?

"Conexión física o conceptual entre objetos." Un objeto colabora con otros objetos a través de sus enlaces con éstos. Un enlace denota la asociación específica por la cual un objeto (cliente) utiliza los servicios de otro objeto (servidor), o a través del cual un objeto puede comunicarse con otro. El enlace es una relación semántica, es una relación de uso entre dos elementos, no hay jerarquía, es una relación de iguales.

95. ¿Cómo pueden ser los mensajes entre dos objetos en una relación de enlace?

El paso de mensajes entre dos objetos es típicamente unidireccional, aunque ocasionalmente puede ser bidireccional.

96. ¿Qué es un mensaje unidireccional?

Cuando un objeto activo invoca a 1 objeto servidor o agente (que actúe como pasivo)

97. ¿Qué es un mensaje bidireccional?

Cuando 1 objeto activo invoca al pasivo y el pasivo le puede devolver datos al objeto activo

98. ¿Quién inicia el paso de un mensaje entre dos objetos en una relación de enlace?

Lo inicia el cliente u objeto activo

99. ¿Cuáles son los roles o papeles que puede desempeñar un objeto en una relación de enlace?

El de Actor, servidor y agente.

100. ¿Qué significa que un objeto actúe como "Actor"?

Un objeto que puede operar sobre otros objetos pero nunca se opera sobre él por parte de otros objetos. Un objeto invoca a un método de otro, inicia la relación de enlace, solo opera sobre otro, invoca a terceros. En algunos contextos, los términos objeto activo y actor son equivalentes.

101. ¿Qué significa que un objeto actúe como "Servidor"?

Un objeto que nunca opera sobre otros objetos, sólo otros objetos operan sobre él. No puede invocar métodos de otros, solo recibe invocaciones.

102. ¿Qué significa que un objeto actúe como "Agente"?

Un objeto que puede operar sobre otros objetos y además otros objetos pueden operar sobre él; un agente se crea normalmente para realizar algún trabajo en nombre de un actor u otro agente. Invoca y recibe métodos de otros, puede ser actor o servidor.

103. Dados dos objetos A y B, si A le puede enviar un mensaje a B, estando ambos relacionados por enlace, decimos que B respecto de A está: ……VISIBLE**……**

104. ¿Cuáles son las cuatro formas de visibilidad que puede poseer un objeto servidor respecto de un objeto cliente?

* El objeto servidor es global para el cliente.
*El objeto servidor es un parámetro de alguna operación del cliente.
* El objeto servidor es parte del objeto cliente.
*El objeto servidor es un objeto declarado localmente en alguna operación del cliente.

105. En una relación de enlace de dos objetos, cuando uno le pasa un mensaje al otro, además de adoptar roles ambos deben estar:……SINCRONIZADOS**……..**

106. ¿Cuáles son las posibles formas de sincronización?

Secuencial, vigilado y síncrono.

107. ¿Qué significa que dados dos objetos A y B estos están secuencialmente sincronizados?

El funcionamiento del objeto pasivo está garantizado por el accionar de un único objeto activo simultáneamente. Hay un solo hilo de ejecución o de atención (uno solo por vez)

108. ¿Qué significa que la forma de sincronizarse de un conjunto de objetos es vigilada?

La semántica del objeto pasivo está garantizada por la utilización de múltiples hilos de control. Los clientes activos deben colaborar para asegurar la exclusión mutua. Hay varios hilos de ejecución simultáneos, varios objetos activos, pero entre ellos se organizan para ver quién es atendido primero.

109. ¿Qué significa que la forma de sincronizarse de un conjunto de objetos es síncrona?

El funcionamiento del objeto pasivo está garantizado por la utilización de múltiples hilos de control. El servidor garantiza la exclusión mutua. Hay varios hilos pero ahora el servidor pone el orden y no el cliente como en el caso anterior.

UNIDAD 2

110. ¿El enlace es una relación de igual a igual o jerárquica?

De igual a igual.

111. ¿La agregación es una relación de igual a igual o jerárquica?

Jerárquica.

112. ¿Qué tipo de jerarquía denota la agregación?

Todo/parte.

113. ¿Qué otros nombres recibe el "todo" en una relación de agregación?

Agregado (todo) o contenedor.

114. ¿En una relación de agregación las "partes" forman parte del estado del "todo"?

Sí

115. ¿Qué tipos de agregación existen?

*Con contención física: Cuando un objeto no existe sin el otro. Sus ciclos de vida están íntimamente relacionados. Por ejemplo un avión se compone de alas, motores, etc (es un caso de contención física)

*Sin contención física: El ciclo de vida de los elementos son independientes uno del otro. Ejemplo: la relación entre un accionista y sus acciones es una relación de agregación que no requiere contención física. El accionista únicamente posee acciones, pero las mismas no son de ninguna manera parte física del accionista.

116. ¿Qué caracteriza a la agregación con contención física?

Cuando un objeto no existe sin el otro. Sus ciclos de vida están íntimamente relacionados. Por ejemplo un avión se compone de alas, motores, etc (es un caso de contención física)

117. ¿Qué es una clase?

Mientras que un objeto es una entidad concreta que existe en el tiempo y el espacio, una clase representa sólo una abstracción, la esencia de un objeto. En términos corrientes, se puede definir una clase como un grupo, conjunto o tipo marcado por atributos comunes o un atributo común, una división, distinción o clasificación de grupos basada en la calidad, grado de competencia o condición.

En el contexto del análisis y diseño orientados a objetos, se define una clase como sigue: "Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común". También se dice que es un molde para los objetos. Las clases al igual que los objetos, no existen aisladamente.

118. ¿La interfaz de la clase proporciona su visión interna?

No. Esto lo proporciona su implementación. La implementación de una clase es su visión interna, que engloba los secretos de su comportamiento. La implementación de una clase se compone principalmente de la implementación de todas las operaciones definidas en la interfaz de la misma.

119. ¿La implementación de la clase proporciona su visión externa?

No, esto lo hace la interfaz (es lo externo, lo que se ve. Se dice que es un contrato. Proporciona su visión externa y por lo tanto, enfatiza la abstracción a la vez que oculta su estructura y los secretos de su comportamiento. La interfaz se compone de las declaraciones de todas las operaciones aplicables a instancias de esta clase, pero también puede incluir la declaración de otras clases, constantes, variables y excepciones, según se necesiten para completar la abstracción).

120. ¿En cuántas partes la podemos dividir una interfaz en términos de la accesibilidad o visibilidad que posee?

*Public: una declaración accesible a todos los clientes.

*Protected: una declaración accesible sólo a la propia clase, sus subclases, y sus clases amigas (friends)

*Private: una declaración accesible sólo a la propia clase y sus clases amigas.

121. ¿Qué tipos básicos de relaciones existen entre las clases?

*Generalización/ especialización: que denota una relación es-un.

*Todo/Parte: que denota una relación parte-de. Así un pétalo no es un tipo de flor, es una parte de la flor.

*Asociación: denota alguna dependencia semántica entre clases de otro modo independientes. Es una relación potencial de objetos que resulten de esa clase. Ejemplo: abeja que poliniza una flor (se puede dar o no). En las clases es estático y se llama asociación, a nivel objeto es dinámico y se llama enlace (se da en tiempo de ejecución.) Otro ejemplo: las rosas y las velas son clases claramente independientes, pero ambas representan cosas que podrían utilizarse para decorar la mesa de una cena.

122. ¿Qué relaciones entre clases se desprenden de las tres relaciones básicas?

*Herencia,
*Asociación (si no la pongo como básica)
* Agregación
* Uso
*Instanciación
*Metaclase.

123. ¿La asociación denota una dependencia semántica y la dirección de esta asociación?

Una asociación sólo denota una dependencia semántica y no establece la dirección de esta dependencia (a menos que se diga lo contrario, una asociación implica relación bidireccional) ni establece la forma exacta en que una clase se relaciona con otra (sólo puede denotarse esta semántica nombrando el papel que desempeña cada clase en relación con la otra).

124. ¿Qué significa la cardinalidad en una relación?

Es la multiplicidad con que se relaciona una clase con otra. Es importante modelar la forma como los objetos se asocian entre sí. Además es necesario identificar el significado de la asociación y la cantidad de objetos con los que un objeto dado puede y debe asociarse.

125. ¿Qué cardinalidad puede existir entre clases relacionadas por asociación?

*Uno a uno
*Uno a muchos
*Muchos a Muchos.

126. ¿Qué es la herencia?

Es una relación jerárquica de clases. Capacidad por la cual una clase de orden inferior puede recibir estructura o acciones de una o más clases de orden superior. La subclase posee la capacidad de incorporar parte estructural y acciones propias.

127. ¿Cuántos tipos de herencia existen?

*Herencia Simple
*Herencia Múltiple

128. ¿A qué se denomina herencia simple?

Es una relación entre clases en la que una clase comparte la estructura y/o comportamiento definidos en una clase (super). La herencia simple se realiza tomando una clase existente y derivando nuevas clases de ella. La clase derivada hereda las estructuras de datos y funciones de la clase original. Además, se pueden añadir nuevos miembros a las clases derivadas y los miembros heredados pueden ser modificados. Una clase utilizada para derivar nuevas clases se denomina clase base, clase padre, superclase o ascendiente La herencia simple es aquella en la que cada clase derivada hereda de una única clase. En herencia simple, cada clase tiene un solo ascendiente. Cada clase puede tener, sin embargo, muchos descendientes.

129. ¿A qué se denomina herencia múltiple?

Es una relación entre clases en la que una clase comparte la estructura y/o comportamiento definidos en varias clases.

130. ¿Cómo se denomina a la clase que no se espera tener instancias de ella y solo se utilizará para heredar a otras clases?

Clase Abstracta. Una clase abstracta se crea con la idea de que las subclases añadan cosas a su estructura y comportamiento, usualmente completando la implementación de sus métodos (habitualmente incompletos)

131. ¿Cómo se denomina a la clase que se espera tener instancias de ella y puede utilizarse para heredar a otras clases o no?

Clase concreta.

132. ¿Cómo se denomina al método de una clase abstracta que no posee implementación y fue pensado para que sea implementado en las sub clases que lo heredan?

Método virtual. Es aquel que no posee implementación y se declara en una clase abstracta o virtual. Cuando este método es heredado en clase sellada o concreta obligatoriamente debe implementarse. Si en lugar de ello la herencia es recibida por otra clase abstracta el método virtual puede o no tener implementación.

133. ¿Cómo se denomina a la clase más generalizada en una estructura de clases?

Superclase o clase base.

134. ¿Qué es el polimorfismo?

Es un concepto de teoría de tipos en el que un nombre puede denotar instancias de muchas clases diferentes en tanto en cuanto estén relacionadas por alguna superclase común. Cualquier objeto denotado por este nombre es, por tanto, capaz de responder a algún conjunto común de operaciones de diversas formas.

Uno de los objetivos principales de las técnicas OO es utilizar otra vez el código. Sin embargo, algunas de las operaciones requieren adaptación para resolver necesidades particulares. Esta necesidad, se da generalmente entre

superclases y subclases, donde una subclase es una especialización de su superclase, y puede requerir alcanzar los mismos objetivos, pero con distintos mecanismos. Por ejemplo, una superclase rectángulo podría tener una operación área cuyo objetivo es calcular el área del rectángulo, definida como la multiplicación de los largos de dos lados contiguos. A su vez, la clase cuadrado es una subclase de rectángulo que también tiene una operación área cuyo objetivo es calcular el área del cuadrado, pero que está definida especialmente para los objetos del tipo cuadrado como la multiplicación del largo de uno de sus lados por sí mismo.

El fenómeno recién descrito se conoce como polimorfismo, y se aplica a una operación que adopta varias formas de implantación según el tipo de objeto, pero cumple siempre el mismo objetivo.

Una de las ventajas del polimorfismo es que se puede hacer una solicitud de una operación sin conocer el método que debe ser llamado. Estos detalles de la implantación quedan ocultos para el usuario; la responsabilidad descansa en el mecanismo de selección de la implantación OO.

"Capacidad por la cual una acción puede responder de distinta forma de acuerdo a la subclase que la implementa."

135. ¿Cómo se denomina cuando una clase posee métodos que comparten el nombre y se diferencias por su firma?

Se denomina Sobrecarga.

136. ¿Qué sentencias de código se evitan utilizar cuando se aplica correctamente el polimorfismo?

El uso de los case y los if anidados lo que hacen al código muy acoplado.

137. ¿Qué es la agregación cómo relación entre clases?

Las relaciones de agregación entre clases tienen un paralelismo directo con las relaciones de agregación entre los objetos correspondientes a esas clases. Relación Jerárquica del tipo Todo – Parte. La agregación es un caso particular de la asociación que denota posesión.

La agregación no precisa contención física. La herencia múltiple no es agregación!.

138. ¿Qué formas de contención física existen en la agregación?

*Con contención Física (se subdivide en valor y referencia)
* Sin contención Física. (el ciclo de vida de los elementos son independientes uno del otro)

139. ¿Qué características posee la contención física por valor?

Significa que el objeto no existe independientemente de la instancia que lo encierra. El tiempo de vida de los objetos está en íntima conexión. La contención por valor no puede ser cíclica (es decir, ambos objetos no pueden ser físicamente partes de otro)

140. ¿Qué características posee la contención física por referencia?

Sus ciclos de vida no están íntimamente relacionados. Se pueden crear y destruir instancias de cada clase independientemente. Este tipo de contención puede ser cíclica (cada objeto puede tener un puntero apuntando al otro)

141. ¿Qué es una relación de uso?

Mientras que una asociación denota una conexión semántica bidireccional, una relación de uso es un posible refinamiento de una asociación, por el que se establece qué abstracción es el cliente y qué abstracción es el servidor que proporciona ciertos servicios. Es una asociación refinada, se especifican roles (actor, cliente, servidor)

142. ¿Qué es la instanciación?

Acción por la cual se crean instancias de una clase. Los objetos creados corresponden al tipo de la clase que los origina. Es la acciono por la cual sometemos a una clase para obtener un objeto.

143. ¿Todo objeto es una instancia de una clase?

Sí.

144. ¿Qué es una metaclase?

Es una clase de una clase. Dicho de otra forma, es una clase cuyas instancias son, ellas mismas.

145. ¿Qué métricas hay que observar para determinar la calidad de una abstracción?

*Acoplamiento
*Cohesión
*Suficiencia
*Compleción
*Ser primitivo

146. ¿Qué es el acoplamiento?

La medida de la fuerza de la asociación establecida por una conexión entre un módulo y otro. El acoplamiento fuerte complica un sistema porque los módulos son más difíciles de comprender, cambiar o corregir por si mismos si están muy interrelacionados con otros módulos. La complejidad puede reducirse diseñando sistemas con los acoplamientos más débiles posibles entre los módulos.

147. ¿Qué es la cohesión?

La cohesión mide el grado de conectividad entre los elementos de un solo modulo (y para el diseño orientado a objetos, una clase u objeto). La forma de cohesión menos deseable es la cohesión por coincidencia, en la que se incluyen en la misma clase o modulo abstracciones sin ninguna relación. La forma más deseable de cohesión es la cohesión funcional, en la cual los elementos de una clase o modulo trabajan todos juntos para proporcionar algún comportamiento bien delimitado.

148. ¿Qué es la suficiencia?

La clase o módulo captura suficientes características de la abstracción como para permitir una interacción significativa y eficiente. Lo contrario produce componentes inútiles. En otras palabras es el grado en que una clase representa elementos suficientes y necesarios para que sea una implementación eficiente.

149. ¿Qué es la compleción?

Por completo, quiere decirse que la interfaz de la clase o módulos captura todas las características significativas de la abstracción. Mientas la suficiencia implica una interfaz mínima, una interfaz completa es aquella que cubre todos los aspectos de la abstracción. Una clase o módulo completo es aquel cuya interfaz es suficientemente general para ser utilizable de forma común por cualquier cliente. Según apuntes de clase: es el grado en que la clase contempla características significativas de la abstracción.

150. ¿Qué significa ser primitivo?

Las operaciones primitivas son aquellas que pueden implantarse eficientemente sólo si tienen acceso a la representación subyacente de la abstracción. Apunte de clase: lo más básica posible para que sea reutilizable. Todo esto depende del contexto.

151. ¿Qué se debe observar al momento de decidir si una abstracción debe implementar un determinado comportamiento o no?

*Reutilización (seria este comportamiento más útil en más de un contexto?)
*Complejidad (que grado de dificultad plantea implementar este comportamiento)
*Aplicabilidad (que relevancia tiene este comportamiento para el tipo en el que podría ubicarse)
*Conocimiento de la implementación (depende del comportamiento de los detalles internos?)

152. ¿Qué formas puede adoptar el paso de un mensaje?

*Síncrono
*Abandono inmediato
*De intervalo
*Asíncrono

153. ¿Qué características posee un mensaje síncrono?

Una operación comienza sólo cuando el emisor ha iniciado la acción y el receptor está preparado para aceptar el mensaje, el emisor y el receptor esperarán indefinidamente hasta que ambas partes estén preparadas para continuar.

154. ¿Qué características posee un mensaje de abandono inmediato?

Igual que el síncrono, excepto en que el emisor abandonará la operación si el receptor no está preparado inmediatamente.

155. ¿Qué características posee un mensaje de intervalo?

Igual que el síncrono excepto en que el emisor esperará a que el receptor esté listo solo durante un intervalo de tiempo especificado.

156. ¿Qué características posee un mensaje Asíncrono?

Un emisor puede iniciar una acción independientemente de si el receptor está esperando o no el mensaje.

157. ¿Qué significa que una abstracción está accesible a otra?

Por accesible, se entiende la capacidad de una abstracción para ver a otra y hacer referencia a recursos en su vista externa. Una abstracción es accesible a otra sólo donde sus ámbitos se superpongan y sólo donde estén garantizados los derechos de acceso (por ejemplo, las partes privadas de una clase son accesibles sólo a la propia clase y sus amigas)

158. ¿Qué expresa la ley de Demeter?

Los métodos de una clase no deberían depender de ninguna manera de la estructura de ninguna clase, salvo de la estructura inmediata (de nivel superior) de su propia clase. Además cada método debería enviar mensajes sólo a objetos pertenecientes a un conjunto muy limitado de clases.

159. ¿Cuál es la consecuencia inmediata de aplicar la ley de Demeter?

El efecto básico de la aplicación de esta ley es la creación de clases débilmente acopladas, cuyos secretos de implantación están encapsulados. Tales clases están claramente libres de sobrecargas, lo que significa que para comprender el significado de una clase no es necesario comprender los detalles de muchas otras clases.

160. ¿Cuáles son las cuatro formas fundamentales por las cuales un objeto X puede hacerse visible a un objeto Y?

*El objeto proveedor es global al cliente.
*El objeto proveedor es parámetro de alguna operación del cliente.
*El objeto proveedor es parte del objeto cliente.
*El objeto proveedor es un objeto declarado localmente en el ámbito del diagrama de objetos.

161. ¿Para qué sirve clasificar a los objetos?

La clasificación es el medio por el que ordenamos el conocimiento. En el diseño orientado a objetos, el reconocimiento de la similitud entre las cosas nos permite exponer lo que tienen en común en abstracciones clave y mecanismos, y eventualmente nos lleva a arquitecturas más pequeñas y simples.

La clasificación es el medio por el cual ordenamos el conocimiento. La clasificación ayuda a identificar jerarquías de generalización, especialización y agregación entre clases. Proporciona también una guía para tomar decisiones sobre modularización, se puede decidir ubicar ciertas clases y objetos juntos en el mismo módulo o en módulos diferentes.

162. ¿Por qué es tan difícil la clasificación de objetos?

Primero porque cualquier clasificación es relativa a la perspectiva del observador que la realiza (hay tantas formas de dividir el mundo en sistemas de objetos como científicos para emprender esa tarea) y segundo, la clasificación inteligente requiere una gran cantidad de imaginación creativa.

163. ¿Cómo es el rol del observador en la clasificación de objetos?

Trascendente, muy importante. El problema central de observación es sin duda el observador, porque debe asimilar la información derivada de sus observaciones y después sacar conclusiones acerca de sus construcciones hipotéticas. Puede hacer inferencias por completo erróneas. Por el contrario, si el observador es por completo objetivo y no conoce el tema de la observación puede que lo observado no sea lo adecuado. La observación exige un conocimiento competente de lo observado y de su significado.

164. ¿Cuáles son las aproximaciones generales a la clasificación?

*Categorización Clásica
*Agrupamiento Conceptual
*Teoría de prototipos

165. ¿Qué es la categorización clásica?

En la categorización clásica, todas las entidades que tienen una determinada propiedad o colección de propiedades en común forman una categoría. Tales propiedades son necesarias y suficientes para definir la categoría. Por ejemplo, las personas casadas constituyen una categoría (se está casado o no) ya que el valor de esta propiedad es suficiente para decidir a qué grupo pertenece determinada persona. Por otra parte, las personas altas no forman una categoría, a menos que pueda haber un acuerdo respecto a algún criterio absoluto por el que se distinga la propiedad alto de la propiedad bajo. Entonces la aproximación clásica a la categorización emplea propiedades relacionadas como criterio de similitud entre objetos. Concretamente se puede dividir los objetos en conjuntos dependiendo de la presencia o ausencia de una propiedad particular.

166. ¿Qué es el agrupamiento conceptual?

Deriva en gran medida de los intentos de explicar cómo se representa el conocimiento. En este enfoque, las clases (agrupaciones de entidades) se generan formulando primero descripciones conceptuales de estas clases y clasificando entonces las entidades de acuerdo con las descripciones. Por ejemplo, se puede establecer un concepto como “una canción de amor”

Resumiendo, se definen pautas descriptivas. Se desarrolla una estructura conceptual. Se agrupan todas aquellas que respondan a la descripción establecida.

167. ¿Qué es la teoría de prototipos?

Se crean clases prototípicas. A todas aquellas que se le aproximan en forma significativa se las considera pertenecientes a ese tipo. Parte de escoger un objeto prototipo que representa a una clase de objetos, y considerar a otros objetos como miembros de la clase si y sólo si se parecen de modo significativo al prototipo.

168. ¿Qué es una abstracción clave?

Es una clase u objeto que forma parte del vocabulario del dominio del problema. El valor principal que tiene la identificación de tales abstracciones es que dan unos límites al problema; enfatizan las cosas que están en el sistema y, por tanto, son relevantes para el diseño, y suprimen las cosas que están fuera del sistema que son superfluas. La identificación de abstracciones clave es altamente específica de cada dominio.

Su identificación requiere descubrimiento e invención. Esta identificación inicial da lugar a abstracciones que, en muchos casos, resultan erróneas. Requieren un proceso de refinado y estructuración jerárquica. Se aconseja seguir las reglas de notación:

*Los objetos deben denominarse con nombres propios, mientras las clases tienen nombres comunes.
* Operaciones modificadoras deben denominarse con verbos activos, mientras las operaciones de selección usan preguntas o nombres con el verbo ser.

169. ¿Qué son los mecanismos?

Un mecanismo describe cualquier estructura mediante la cual los objetos colaboran para proporcionar algún comportamiento que satisface un requerimiento del problema. Mientras el diseño de una clase incorpora el conocimiento de cómo se comportan los objetos individuales, un mecanismo es una decisión de diseño sobre cómo cooperan colecciones de objetos. Los mecanismos representan así patrones de comportamiento.

Los mecanismos son los medios por los cuales los objetos colaboran para proporcionar algún comportamiento de nivel superior. Los mecanismos denotan decisiones estratégicas de diseño respecto a la actividad de colaboración entre muchos tipos diferentes de objetos. Los mecanismos permiten considerar el trabajo conjunto de las clases para producir un comportamiento complejo.

UNIDAD 3

170. ¿Qué es un Framework?

"Un conjunto de clases que incorporan un diseño abstracto para soluciones a una familia de problemas relacionados, y soporta re-uso a un nivel de granularidad mayor que el de las clases"

Es un marco de trabajo que ofrece a quien lo utiliza, una serie de herramientas para facilitarle la realización de tareas. Puede contener librerías de clases, documentación, ayuda, ejemplos, tutoriales pero sobre todo contiene experiencia sobre un dominio de problema específico. Resuelve cuestiones que son comunes a varios sistemas que tienen ciertas similitudes. Es una solución para un entorno de trabajo.

Características principales:

1- Normalizar: homogeneizar para usarlo en forma común y colocarlo en el framework

2- Estandarizar: generar documentación

3- Reutilizar experiencia

171. ¿Qué son los frozen-spots en un Framework?

Algunas de las características del framework no son mutables ni tampoco pueden ser alteradas fácilmente. Estos puntos inmutables constituyen el núcleo o kernel de un framework, también llamados como los puntos congelados o frozen-spots del framework. A diferencia de los puntos calientes o hot-spots, los puntos congelados o inmutables son los fragmentos del código puestos en ejecución ya dentro del framework que llaman a uno o más puntos calientes proporcionados por el ejecutor. Según apunte de clase: son los puntos de entrada de la aplicación hacia el framework en que la aplicación hace uso.

172. ¿Qué son los hot-spots en un Framework?

Los puntos calientes o Hot-spots son las clases o los métodos abstractos que deben ser implementados o puestos en ejecución. Es el servicio que presta el framework que requiere especificación de detalle a nivel de aplicación.

173. ¿Cómo se puede clasificar un Framework según su extensibilidad?

*De Caja Negra
*De Caja Blanca

174. ¿Qué es un Framework de Caja Blanca?

Para su instanciación, el desarrollador necesita conocer su estructura interna. Al utilizarlos, el mecanismo predominante es la herencia, mediante la que se hacen concretas las propiedades abstractas de las clases del framework.

175. ¿Qué es un Framework de Caja Negra?

En un framework caja negra, existen muchas alternativas implementadas para los hot spots y su uso se reduce (aunque no se limita), a configurar el framework eligiendo la combinación adecuada al caso. Es importante destacar que aún con pocas implementaciones de hot spots pueden conseguirse muchas aplicaciones distintas, ya que estas dependen del número de combinaciones.

Los frameworks de caja negra producen instancias usando escrituras o scripts de configuración. Después de la configuración, una herramienta automática de instanciación crea las clases y el código de fuente (Source Code). La caja negra del framework no requiere que el usuario conozca los detalles internos del mismo. Para su instanciación no es preciso conocer la forma en que están construidos. El desarrollador con personaliza los puntos calientes mediante la instanciación con parámetros actuales y la composición.

176. ¿Qué ventajas posee utilizar un Framework?

•Permite abstraerse de problemas resueltos con amplia experiencia previa.
•En el caso de.NET el programador no piensa en arquitectura sino en programas.
•Permite aprovechar todas las ventajas de la estandarización.
*Las aplicaciones crecen y necesitan mantenimiento. Los frameworks facilitan la separación de la presentación y la lógica además de mantener una sintaxis coherente.

177. ¿Qué problemas resuelve .NET Framework?

•Desde Internet, muchas aplicaciones y dispositivos están fuertemente comunicados entre sí.
•Los programadores escribían arquitectura en lugar de aplicaciones.
•Los programadores tenían conocimientos limitados o debían aprender nuevos lenguajes.
•El .NETFramework constituye las bases sobre las que, tanto aplicaciones como servicios, son ejecutadas y construidas.
•La naturaleza unificada del .NET Framework permite que cualquier tipo de aplicación sea desarrollada mediante herramientas comunes haciendo la integración mucho más simple.

178. ¿Qué es y qué permite hacer el CLR?

•El CLR es el motor de ejecución (runtime) del.NET Framework.
•Ofrece servicios automáticos tales como:
•Administración de la memoria
•Seguridad del código, asegurando:
•Conversión de tipos
•Inicialización de variables
•Indexación de arreglos fuera de sus límites
•Versionamiento

179. ¿Qué es el MSIL?

A diferencia de lo que sucede en los lenguajes de programación tradicionales, los compiladores .NET no producen código nativo que pueda inyectarse directamente en la CPU y ser ejecutado por ésta. En su lugar, producen el denominado código en Lenguaje Intermedio de Microsoft (MSIL o IL) que es una especia de lenguaje máquina asociado con un procesador virtual que no se corresponde con ninguna CPU. Es de un nivel superior al puro lenguaje ensamblador, tiene en cuenta conceptos de alto nivel tales como excepciones y creaciones de objetos.

180. ¿Qué es el CTS?

Define un conjunto común de “tipos” orientado a objetos.

•Todo lenguaje de programación debe implementar los tipos definidos por el CTS.

•Todo tipo hereda directa o indirectamente del tipo OBJECT

•Tipos de VALOR y de REFERENCIA

181. ¿Qué es el CLS?

La interoperabilidad entre lenguajes es la posibilidad de que el código interactúe con código escrito en un lenguaje de programación diferente. La interoperabilidad entre lenguajes puede ayudar a maximizar la reutilización de código y, por tanto, puede mejorar la eficacia del proceso de programación. Dado que los desarrolladores utilizan una gran variedad de herramientas y tecnologías, cada una de las cuales podría admitir distintos tipos y características, desde tiempo atrás ha sido complicado garantizar la interoperabilidad entre lenguajes. No obstante, los compiladores y las herramientas de lenguaje dirigidos a Common Language Runtime se benefician de la compatibilidad que integra el motor en tiempo de ejecución para la interoperabilidad entre lenguajes. Para garantizar que el código administrado será accesible para los desarrolladores que empleen otros lenguajes de programación, .NET Framework proporciona Common Language Specification (CLS), que describe un conjunto fundamental de características de lenguaje y define reglas sobre cómo usar dichas características.

UNIDAD 4

182. ¿Qué extensión poseen los archivos en VB.NET?

VB.NET utiliza la extensión de archivo para determinar el lenguaje que cada archivo contiene; por lo tanto los archivos de Visual Basic. NET tienen la extensión .vb

183. ¿Qué diferencia existe en .NET entre un archivo de módulo y un archivo de clase?

El motivo de que VB.NET utilice una única extensión para todos sus archivos es que no existen muchas diferencias entre un archivo módulo o un archivo de clase. El concepto de archivo de módulo o archivo de clase ha desaparecido en VB.NET ya que un archivo puede contener varios módulos y varias clases o una combinación de ambos.

184. ¿Cuáles son las palabras claves para definir un módulo?

Module (nombre del módulo)– End Module

185. Mencionar al menos tres cosas que puede contener un módulo.

Los módulos trabajan como contenedores de procedimientos y variables.

1. Variables
2. Funciones
3. Procedimientos (sub)

186. ¿Qué visibilidad posee un elemento público colocado en un módulo?

Es visible en todo el proyecto

187. ¿Qué visibilidad posee un elemento privado colocado en un módulo?

Es solo visible para el código contenido dentro del bloque

188. ¿Un módulo puede tener un procedimiento Sub New?

Sí puede. Si un módulo contiene un procedimiento SubNew, la rutina llamará a este procedimiento antes de ejecutar el código contenido en el propio módulo.

189. ¿Un módulo puede tener más de un procedimiento Sub Main?

No, en el mismo módulo no es posible.

190. ¿Qué característica posee el procedimiento Sub New?

Si un módulo contiene un procedimiento SubNew, la rutina llamará a este procedimiento antes de ejecutar el código contenido en el propio módulo. El runtime llama al procedimiento SubNew la primera vez que el proyecto que se encuentra en ejecución haga referencia a una variable o procedimiento del módulo**.** El procedimiento SubNew no se ejecutará cuando se acceda a una constante definida en el módulo.

191. ¿Qué característica posee el procedimiento Sub Main?

El Sub Main es el punto de entrada a la aplicación. Con VB.NET se puede seleccionar el módulo que contenga un procedimiento Sub Main que se ejecutará cuando el programa se inicie. Para realizar esto se debe utilizar el cuadro de diálogo Páginas de propiedades del proyecto. Con este módulo se puede lograr que una aplicación comience a ejecutar el código contenido en él, si se le especifica al proyecto en sus propiedades. Pueden coexistir varios Sub Main en el proyecto.

192. ¿Qué se ejecuta primero en un módulo si este posee Sub Main y Sub New?

Sub New

193. ¿Cómo se declara una clase en un módulo?

Class End Class.

194. ¿Qué cosas comunes podemos encontrar en una clase?

*Una clase puede contener variables Private o Public (o campos en la terminología de .NET)
*Puede contener funciones y procedimientos (métodos) Private o Public.

195. ¿Cómo se construye un formulario en .NET?

En .NET los formularios no son más que clases.
Las clases que representan formularios heredan de System.Windows.Forms.Form.

196. ¿Qué posee .NET para trabajar los aspectos visuales de un formulario?

Existe un diseñador de formularios para desarrollar el aspecto visual de los formularios.

197. ¿Qué procedimiento utiliza y necesita fundamentalmente el diseñador de formularios?

Private sub InitializeComponent()

198. ¿Cuál debería ser la primera instrucción en el constructor de un formulario?

El formulario posee un constructor. La primer instrucción debería ser la que invoca al constructor de la clase base(MyBase.New).

199. ¿Qué es un NameSpace?

Los Espacios de Nombres (NameSpaces) son bloques que permiten agrupar lógicamente los siguientes elementos: Module,Class,Structure,Interface yEnum. Todo proyecto posee un NameSpace por defecto denominado Espacio de Nombres Raíz. Se pueden tener varios NameSpace con el mismo nombre en el mismo proyecto, esto hará que su contenido se muestre todo junto.

200. ¿Cómo se define un NameSpace?

Namespace (Nombre)…. End NameSpace

201. ¿Qué se puede colocar dentro de un NameSpace?

*Module
*Class
*Structure
*Interface
*Enum.

Muy importante: no se puede definir directamente declaraciones de variables o procedimientos dentro de un bloque Namespace.

202. ¿Con qué instrucción se puede poner disponible en un proyecto el contenido de un NameSpace?

La instrucción Imports sirve para indicarle al compilador que el código contenido en el archivo fuente puede acceder a todas las clases, procedimientos y estructuras definidas en un determinado espacio de nombres.
Sintaxis:ImportsSeresVivos.Animales

203. ¿Se pueden anidar los NameSpace?

Si, es posible anidarlos.

204. ¿La instrucción Imports se puede colocar en cualquier lado? Justifique

La instrucción Imports se debe colocar precediendo cualquier otra en un Módulo.

205. ¿Qué ámbitos pueden tener las variables?

Dim–Private–Public–Static

206. ¿Qué visibilidad posee Dim a nivel de procedimiento?

Se deberá utilizar Dim dentro de un procedimiento para declarar una variable local (dinámica), que sólo será visible dentro de dicho procedimiento.

207. ¿Qué visibilidad posee Dim a nivel de clase?

Se deberá utilizar Dim o Private fuera de los bloques de procedimiento, pero dentro de un bloque Class o Module, para crear variables a las que pueda acceder desde cualquier lugar dentro de la clase o del módulo pero no desde cualquier lugar del proyecto (sin embargo, la instrucción DIM dentro de una estructura tendrá un ámbito Public)

208. ¿Qué visibilidad posee Public a nivel de módulo?

Global. Utilizando Public:Dentro de un bloque Module para que posean visibilidad en todo el proyecto o dentro de un bloque Class para que posea visibilidad a nivel de clase.

209. ¿Qué efecto provoca colocarle ámbito de Static a una variable?

Se utiliza a nivel de variable en un procedimiento o función para transformarla en estática.

210. ¿Para qué sirve Option Explicit?

Obliga a la declaración explícita de las variables. Acepta valores On y Off.

211. ¿Para qué sirve Option Compare?

Declara el método de comparación predeterminado que se utilizará al comparar los datos de tipo cadena.

212. ¿Qué valores acepta Option Compare?

Binary y Text

213. ¿Qué efecto posee el valor Binary en el Option Compare?

Se producirán comparaciones de cadenas basadas en un tipo de ordenación que se deriva de las representaciones binarias internas de los caracteres.

214. ¿Qué efecto posee el valor Text en el Option Compare?

Se producirán comparaciones de cadenas basadas en un texto que utiliza un orden que no distingue entre mayúsculas y minúsculas y que se basa en la configuración de idioma del sistema.

215. ¿Para qué se utiliza Option Strict?

Restringe las conversiones de tipos de datos implícitas únicamente a conversiones de ampliación (por ejemplo de single a doublé, no pierde precisión)

216. ¿Para qué se utiliza Option Infer?

El compilador de VisualBasic2008 utiliza la inferencia de tipos para determinar los tipos de datos de las variables locales declaradas sin ninguna cláusula As. El compilador deduce el tipo de la variable a partir del tipo de la expresión de inicialización. De esta forma, puede declarar las variables sin indicar explícitamente un tipo.

217. Dar un ejemplo de uso de Option Infer.

```
Public Sub inferenceExample()
'Tipado explicito.
        Dim num1 As Integer=3
'Usa tipado por inferencia
        Dim num2 = 3
End Sub
```

218. ¿El siguiente código es correcto? Justifique

```
Sub Prueba()
Dim x As Integer
Dim Y As Integer = 10
For x = 1 To 10
Dim y As Integer = 20
Next
End sub
```

No es correcto porque la variable a nivel procedimiento tiene el mismo nombre que la variable a nivel de bloque.

219. ¿Qué tipos de datos son de punto flotante?

Double y Decimal

220. ¿Cuántos bytes ocupa un tipo Boolean?

4Bytes

221. ¿Cuántos bytes ocupa un tipo Byte?

1Byte

222. ¿Cuántos bytes ocupa un tipo Char?

2Bytes

223. ¿Cuántos bytes ocupa un tipo Date?

8Bytes

224. ¿Cuántos bytes ocupa un tipo Decimal?

12Bytes

225. ¿Cuántos bytes ocupa un tipo Double?

8Bytes

226. ¿Cuántos bytes ocupa un tipo Integer?

4Bytes

227. ¿Cuántos bytes ocupa un tipo Long?

8Bytes

228. ¿Cuántos bytes ocupa un tipo Object?

4Bytes

229. ¿Cuántos bytes ocupa un tipo Short?

2Bytes

230. ¿Cuántos bytes ocupa un tipo Single?

4Bytes

231. ¿Cuántos bytes ocupa un tipo String?

10Bytes + (2* Longitud de cadena)

232. ¿Cuántos bytes ocupa un tipo definido por el usuario?

La suma del tamaño de sus miembros

233. ¿Cuántos bytes ocupa un tipo UInteger?

4Bytes

234. ¿Cuántos bytes ocupa un tipo ULong?

8Bytes

235. ¿Cuántos bytes ocupa un tipo UShort?

2Bytes

236. ¿Con qué rango de valores trabaja un tipo Boolean?

True o False

237. ¿Con qué rango de valores trabaja un tipo Byte?

0 a 255 (sin signo)

238. ¿Con qué rango de valores trabaja un tipo Char?

0 a 65535 (sin signo)

239. ¿Con qué rango de valores trabaja un tipo Date?

1 de Enero del 1DC al 31 de diciembre de 9999

240. ¿Con qué rango de valores trabaja un tipo Decimal?

+/- 79.228.162.514.264.337.593.950.335 sin puntos decimales;
+/- 7,9228162514264337593543950335 con 28 decimales a continuación de la coma; el menor valor no cero es +/- ±0,0000000000000000000000000001.

241. ¿Con qué rango de valores trabaja un tipo Double?

-1,79769313486232E308 a 1,79769313486232E308

242. ¿Con qué rango de valores trabaja un tipo Integer?

2.147.483.648 a 2.147.483.647.

243. ¿Con qué rango de valores trabaja un tipo Long?

-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.

244. ¿Con qué rango de valores trabaja un tipo Object?

Se puede almacenar cualquier tipo en una variable tipo Object.

245. ¿Con qué rango de valores trabaja un tipo Short?

-32.768 a 32.767

246. ¿Con qué rango de valores trabaja un tipo Single?

3,402823E38 a 3,402823E38

247. ¿Con qué rango de valores trabaja un tipo String?

0 a aproximadamente 2 millones de caracteres Unicode.

248. ¿Con qué rango de valores trabaja un tipo UInteger?

0 a 4.294.967.295 (sin signo).

249. ¿Con qué rango de valores trabaja un tipo ULong?

0 a 18.446.744.073.709.551.615 (sin signo).

250. ¿Con qué rango de valores trabaja un tipo UShort?

0 to65.535 (sin signo).

251. ¿Se puede declarar una variable e inicializarla en la misma línea?

Si se puede declarar variables en la misma instrucción Dim, Private o Public.

252. ¿Cómo se declara e inicializa una variable en la misma línea?

Dim Num1 As Integer =10

253. ¿Se pueden inicializar variables en la misma línea con datos que no sean constantes?

Si se puede.

254. Dar un ejemplo donde se observe como se declara e inicializa una variable en la misma línea con datos que no sean constantes.

Dim Num1 As Integer

255. ¿Se puede declarar varias variables del mismo tipo en una línea?

Si se puede.

256. ¿Cómo se declaran varias variables del mismo tipo en una línea?

Dim num1,num2,num3 As Integer

257. ¿Se puede declarar varias variables del mismo tipo en una línea e inicializar la última?

No es posible realizar esto.

258. ¿Cómo declaro una variable del tipo cliente y le asigno una instancia de cliente todo en la misma línea de código?

Dim miCliente as Cliente = new Cliente

259. ¿Qué operador se utiliza para saber si la variable C1 apunta al mismo lugar que la variable C2?

Is (devuelve True si apuntan al mismo lugar)

Ejemplo:
Dim C1 as new Auto
Dim C2 as new Auto
C1=C2
Console.writeLine(C1 is C2) → Esto devuelve true.

260. ¿Cuáles son los dos tipos en .NET que permiten clasificar a todos los demás?

Object

261. ¿Dónde se encuentran (dentro de la memoria) los objetos reales que son apuntados por los Reference Type?

Los tipos por referencia (ReferenceType) hacen que una variable puntero apunte al objeto real, el cual se encuentra en el ManagedHeap.

262. ¿Qué tipos de datos son afectados por el GC?

Los tipos por valor se comportan como tipos escalares. Es la misma variable quien almacena el valor. Casi todos los tipos numéricos son ValueType. Los Value Type al no estar en el Managed Heap no se ven afectados por el GC.

263. ¿Cuáles son los operadores de conversión más utilizados?

*Ctype
*DirectCast
*TryCast

264. ¿Qué operador de conversión posee la característica de trabajar tanto con Reference Type como con Value Type y si da un error dispara una excepción?

CType.

Devuelve el resultado de convertir explícitamente una expresión a un tipo de datos, objeto, estructura, clase o interfaz. Funciona con Reference Type oValue Type. Si el argumento no se puede convertir al tipo destino, CType provoca un objeto InvalidCastException.

Dim MiPersona as Persona=Ctype(Objeto,Persona)
Dim Valor as Integer=Ctype(“123.45”,Integer)
Console.WriteLine(Valor)=>123

265. ¿Qué es Boxing y UnBoxing?

Cuando se convierte un Value Type en un Reference Type se desarrolla una operación llamada “Boxing”.
Cuando se convierte un Reference Type a un Value Type se desarrolla una operación llamada “UnBoxing”.
Las operaciones de Boxing y UnBoxing son muy caras en términos de procesamiento. Una buena planificación del tipo de datos que se utilizará minimiza la necesidad que se produzcan operaciones de Boxing y Unboxing.

266. ¿Qué particularidad presenta TryCast respecto Ctype y DirectCast?

Direct Cast: sólo funciona con argumentos de tipos por Referencia, los intentos de pasar tipos de valor se convierten en errores de compilación (CType funciona para ambos tipos). CType siempre intenta convertir el argumento al tipo destino y, por lo tanto, es capaz de convertir una cadena en un tipo numérico, DirectCast sólo funciona si el argumento origen se puede convertir en el tipo destino y provocará InvalidCastException en caso contrario (DirectCast tampoco hace conversiones por ampliación). DirectCast es más rápido que CType. DirectCast se usa en las siguientes Ocasiones:

1. Operación de unboxing sobre un tipo de valor que fuera previamente boxed dentro de una variable object
2. Para convertir de una clase base a una variable de una clase derivada (Por ej: una variable persona a una variable Empleado)
3. Para convertir una variable objeto a una variable de interfaz

TryCast: Introduce una operación de conversión de tipos que no produce una excepción. Si se produce un error al intentar llevar a cabo una conversión, CType y DirectCast producen un error InvalidCastException. Esto puede afectar negativamente al rendimiento de la aplicación. TryCast devuelve Nothing, de modo que en lugar de controlar una posible excepción, sólo debe comprobar el resultado devuelto con Nothing.

267. Mencione al menos seis funciones de conversión de tipos

*CBool
*CInt
*CChar
*CDate
*CDec
*CDble

268. ¿Para qué se utiliza CUInt?

Convierte a tipo UInteger

269. ¿Para qué se utiliza CSng?

Convierte a tipo Single

270. ¿Las constantes pueden ser públicas?

Las constantes pueden ser públicas o privadas. Ej: Public Const MiConstante As String= “Mariano”

271. ¿Las enumeraciones soportan ámbitos?

Las Enumeraciones son bloque de código que facilitan la asignación de valores al ordenarlos. Aceptan ámbitos.Los bloques enum pueden aparecer en cualquier lugar dentro de un archivo fuente, un módulo interno, una clase, bloques o directamente a nivel de espacio de nombres. Ejemplo

```
PrivateEnumPermisos3
Lectura = 10
Escritura = 20
LecturaEscritura= 30
EndEnum
```

272. ¿Las matrices pueden tener como índice base un cero a pesar que por defecto toman este valor?

Las matrices deben tener índice base 0.

273. ¿Qué significa que una matriz es dinámica?

Que no tiene un tamaño fijo definido, va incrementando o decrementando a medida que coloco/saco elementos de la misma. La asignación de memoria también es dinámica.

274. ¿Las matrices trabajan por referencia o por valor al asignar una matriz A a una matriz B?

Por referencia.

275. ¿Qué ocurre si hago MiMatriz1 = MiMatriz2 y cambio un valor en MiMatriz1? Justifique

Como la asignación es por referencia, el valor cambiado en la Matriz1 también cambiará en la Matriz2.

276. ¿Qué ocurre si hago MiMatriz1 = MiMatriz2 y cambio un valor en MiMatriz2? Justifique

Lo mismo que el caso anterior, se modificará ese valor.

277. ¿Qué utilizo si deseo asignar una matriz a otra y que no queden apuntando al mismo objeto en memoria?

El método Clone. Ej Matriz1= Matriz2.Clone

278. ¿Cómo se define un bloque de código para una estructura?

Structure (nombre)… End Structure

279. Mencione al menos tres elementos que puedan ser colocados dentro de una estructura.

*Procedimientos
*Propiedades

280. ¿Cómo haría para que la estructura E1 herede de la estructura E2?

281. ¿De dónde heredan las clases el método Finalize?

Hereda de System.Object

282. ¿Cuál es la firma que implementa el método "Finalize"?

Protected Overrides Sub Finalize ()

283. ¿Cómo obtengo el método "Dispose"?

Implementando la interfaz Idisposable

284. ¿Qué se programa en el método "Dispose"?

Se programa el código de liberación de recursos no administrados.

285. ¿Se pueden combinar el uso de "Dispose" y "Finalize"?

Si, se pueden combinar.

286. ¿Cómo se obtiene el número de generación actual de un objeto?

Através del método GC.getGeneration()

287. ¿Qué utiliza para convertir un objeto en "no" válido para la recolección de elementos no utilizados desde el principio de la rutina actual hasta el momento en que se llamó a este método?

GC.SupressFinalize()

288. ¿Cómo se solicita que el sistema no llame al finalizador del objeto especificado?

GC.SupressFinalize()

289. ¿Qué es una excepción?

Son condiciones no esperadas durante la ejecución de la aplicación o por código ejecutándose dentro de ésta. Es un comportamiento no deseado. Cuando se produce esta situación, se dice que el código inicia una excepción que se espera que otro código sea capaz de capturar.

290. ¿Qué se coloca en el bloque "Catch"?

El código para el tratamiento de la excepción producida en el bloque try.

291. ¿Cómo construiría un objeto del tipo "Exception" personalizado?

Class MiExcepcionPersonalizaException Inherits System.ApplicationException

292. ¿Qué es una excepción?

Son condiciones no esperadas durante la ejecución de la aplicación o por código ejecutándose dentro de ésta. Es un comportamiento no deseado. Cuando se produce esta situación, se dice que el código inicia una excepción que se espera que otro código sea capaz de capturar.

293. ¿Qué ocurre si en el bloque de código donde se produce la excepción el error no está siendo tratado?

La atrapa el framework. Igualmente la excepción se propaga hasta llegar al nivel más superior.

294. ¿Cuál es el objeto de mayor jerarquía para el manejo de excepciones?

Exception

295. ¿En qué namespace se encuentra la clase Exception?

System.Exception

296. ¿Cuáles son las dos clases genéricas más importantes definidas en el FrameWork además de Exception?

*System.System.Exception (la mayoría los obj. Exception definidos en .NET Framework heredan de aquí)

*System.AplicationException (los obj de excepción personalizados y específicos de cada aplicación heredan de aquí)

297. ¿Qué instrucción se utiliza para poner en práctica el control e interceptar las excepciones?

Try
Catch
End Try

298. ¿Dónde se coloca el código protegido contra excepciones si se iniciara una excepción?

Try/ End Try

299. ¿Qué se coloca en el bloque "Catch"?

El código para el tratamiento de la excepción producida en el bloque try.

300. ¿Qué tipo de excepción se utiliza para interceptar un error de división por cero?

DivideByzeroExcepction

301. ¿Qué tipo de excepción se utiliza para interceptar una DLL que tiene problemas al ser cargada?

System.DLLNotFoundException

302. ¿Qué colocaría dentro de una clausula "Catch" para especificar una condición adicional que el bloque "Catch" deberá evaluar como verdadero para que sea seleccionada?

When

303. ¿Si se desea colocar código de limpieza y liberación de recursos para que se ejecute cuando una excepción se produzca, dónde lo colocaría?

Dentro del bloque Finally porque siempre se ejecuta.

304. ¿Qué instrucción se utiliza para provocar un error y que el mismo se adapte al mecanismo de control de excepciones?

Throw. Produce una excepción dentro de un procedimiento. Una instrucción Throw sin expresiones sólo se puede utilizar en una instrucción Catch, en este caso, la instrucción vuelve a producir la excepción que controla la instrucción Catch.

305. ¿Escriba el código que permitiría provocar una excepción del tipo "ArithmeticException"?

```
Class ExceptionTestClass
  Public Shared Sub Main()
    Dim x As Integer = 0
    Try
      Dim y As Integer = 100 / x
    Catch e As ArithmeticException
      Console.WriteLine("ArithmeticException Handler: {0}", e.ToString())
    Catch e As Exception
      Console.WriteLine("Generic Exception Handler: {0}", e.ToString())
    End Try
  End Sub 'Main
End Class 'ExceptionTestClass
```

306. ¿Cómo construiría un objeto del tipo "Exception" personalizado?

Class MiExcepcionPersonalizaException Inherits System.ApplicationException

307. ¿Cómo armaría un "Catch" personalizado para que se ejecute cuando se dé la excepción "ClienteNoExisteException"?

Try

Catch ex As ClienteNoExisteException
Console.WriteLine(ex.Message)

End Try

UNIDAD 5

308. ¿Qué es un Campo de una clase?

Es una propiedad o atributo.

309. ¿Qué es un método de una clase?

Es un procedimiento.

310. ¿A qué denominamos sobrecarga?

Sobrecargar un método significa tener en la misma clase varios métodos con el mismo nombre pero con diferentes firmas de parámetros (con un número distinto de parámetros o con parámetros de diferentes tipos). Recordar colocarle la palabra reservada “Overloads”, es opcional, pero si lo pongo en un método hay que hacerlo en todos.

Con la sobrecarga el código resulta más inteligible y fácil de mantener, también es más eficiente (porque el compilador toma sus decisiones en tiempo de compilación) y robusto (porque las llamadas que no sean válidas no se compilarán). Con la sobrecarga de métodos también evito los if anidados.

311. ¿Qué es una propiedad de una clase?

Los atributos que posee.

312. ¿Qué tipos de propiedades existen?

* Solo Lectura (ReadOnly Property Edad)
* Solo Escritura (WriteOnly Property Edad)
* Lectura/Escritura (Por defecto)
*Propiedades con Argumentos

313. ¿Qué ámbitos pueden tener los campos, métodos y propiedades de las clases?

*Public
*Private
*Friend

314. ¿Qué características posee cada ámbito existente si se lo aplico a un campo, un método y una propiedad de una clase?

Public: Es visto por todas las clases
Private: Es visto por la propia clase
Friend: Por la propia clase + las clases amigas

315. ¿Para qué se utilizan los constructores?

Es un método que se ejecutan cuando se crea una nueva instancia de la clase. Sirven para inicializar el estado de un objeto. Se usan para crear la contención física. Se crea siempre con SubNew()

316. ¿A qué se denomina tiempo de vida de un objeto?

El ciclo de vida de un objeto se extiende desde el momento en que se crea por primera vez (y consume así espacio por primera vez) hasta que ese espacio se recupera. Para crear explícitamente un objeto, hay que declararlo o bien asignarle memoria dinámicamente.

317. ¿Para administrar las instancias de .NET se utiliza un contador de referencias?

No.

318. ¿Qué objeto es el encargado de liberar el espacio ocupado por objetos que ya no se utilizan?

Garbagge Collection.

319. ¿Dónde se encuentran las instancias de los objetos administrados por el GC?

En el Managed Heap, o montón administrado.

320. ¿Cuáles son los dos métodos más notorios que deben implementar las clases para trabajar correctamente con la recolección de elementos no utilizados y matar las instancias administradas y no administradas?

Finalize() y Dispose().

321. ¿De dónde heredan las clases el método Finalize?

System.Object

322. ¿Cuál es la firma que implementa el método “Finalize”?

Finalize()

323. ¿Qué método se utiliza para que el GC recolecte los elementos no utilizados?

GC.Collect()

324. ¿Qué método se utiliza para suspender el subproceso actual hasta que el subproceso que se está procesando en la cola de finalizadores vacíe dicha cola?

GC.WaitForPendingFinalizers()

325. ¿Cuándo se ejecuta el método collect del GC que método se ejecuta en los objetos afectados?

Finalize()

326. ¿Qué método debería exponer una clase bien diseñada teniendo en consideración que no posee destructor?

Debería implementar la interfaz IDisposable, que tiene el método Dispose.

327. ¿Cómo obtengo el método "Dispose"?

```
Class xx Implements IDisposable
Sub Dispose() Implements IDisposable.Dispose
'Cerrar archives y libero recursos
End Sub
End Class
```

328. ¿Qué se programa en el método "Dispose"?

El código para limpieza/ liberación de recursos.

329. ¿Se pueden combinar el uso de "Dispose" y "Finalize"?

Sí, resulta una buena práctica llamar al método Dispose desde dentro del método Finalize porque el código de limpieza suele ser el mismo.

330. ¿A qué se denomina "Resurrección de Objetos"?

Un objeto que haya sido finalizado puede almacenar una referencia de sí mismo en una variable definida fuera de la clase para que esta nueva referencia mantenga vivo al objeto.

331. ¿A qué se denomina "Generación" en el contexto de la recolección de elementos no utilizados?

El recolector de elementos no utilizados cuenta con un método para determinar la edad de un objeto. Cada objeto mantiene un contador que dice el número de recolecciones de elementos no utilizados a las que el objeto ha sobrevivido. El valor de este contador es la generación del objeto. Cuando más elevado sea este número, menor será la posibilidad de que dicho objeto sea eliminado durante la siguiente recolección de elementos no utilizados.

332. ¿Qué valores puede adoptar la "Generación" de un objeto?

Hay sólo tres valores distintos de generación. El valor de la generación de un objeto que nunca ha sufrido una recolección de elementos no utilizados es 0; si el objeto sobrevive a una recolección de elementos no utilizados su generación vale 1; si sobrevive a una segunda recolección de elementos no utilizados, su generación valdrá 2. Cualquier posterior recolección de elementos no utilizados hace que el contador de generación siga valiendo 2 (o se destruye el objeto).

333. ¿Cómo se puede obtener el número de veces que se ha producido la recolección de elementos no utilizados para la generación de objetos especificada?

GC.Collect(0).Como parámetro le paso el número de generación que quiero verificar.

334. ¿Cómo se obtiene el número de generación actual de un objeto?

GC.GetGeneration()

335. ¿Cómo se puede recuperar el número de bytes que se considera que están asignados en la actualidad?

336. ¿Qué utiliza para convertir un objeto en "no" válido para la recolección de elementos no utilizados desde el principio de la rutina actual hasta el momento en que se llamó a este método?

GC.SupressFinalize()

337. ¿Cómo se solicita que el sistema no llame al finalizador del objeto especificado?

GC.SupressFinalize()

338. ¿Cómo se solicita que el sistema llame al finalizador del objeto especificado, para el que previamente se ha llamado a "SuppressFinalize"?

GC.WaitForPendingFinalizers()

339. ¿Cómo se obtiene el número máximo de generaciones que el sistema admite en la actualidad?

GC.GetGeneration()

340. ¿Qué son los sucesos?

Las acciones se construyen a partir de funciones y procedimientos. Los sucesos son reacciones.

341. ¿Qué se utiliza para declarar un suceso?

Event (nombre)

342. ¿Cómo se logra que ocurra un suceso?

RaiseEvent (nombre)

343. ¿Cómo se pueden atrapar los sucesos?

WithEvents

344. ¿para qué se utiliza Addhandler en un suceso?

Permite decidir en tiempo de ejecución que rutina deberá dar servicio a un determinado suceso. AddHandler acepta 2 argumentos: el suceso que voy a redirigir a un procedimiento de la aplicación (en el formato objeto.nombresuceso) y la dirección de la rutina que controla el suceso (en el formato AddressOf nombrerutina).

345. ¿Cómo y qué cosas se pueden compartir en una clase?

Las clases de VB.NET permiten el empleo de campos, propiedades y métodos compartidos. Los miembros compartidos también reciben el nombre de miembros de clase o estáticos en otros lenguajes orientados a objetos.

346. ¿Qué características poseen los campos compartidos?

Los campos compartidos son variables que pueden ser accedidos (es decir, compartidos) por todas las instancias de una clase determinada. Hay que ponerle la palabra Shared adelante. Ej: Shared Nombre as String. Los campos compartidos también resultan de utilidad cuando todas las instancias de una clase compiten entre sí por un número limitado de recursos.

347. ¿Qué características poseen los métodos compartidos?

Se utiliza la palabra Shared para marcar un método como estático, lo que permitirá llamar al método sin necesidad de generar una instancia del objeto de dicha clase.

348. ¿Qué características poseen los sucesos compartidos?

Los sucesos compartidos se definen de la misma forma en que se definen campos, métodos y propiedades compartidas. Se puede provocar sucesos compartidos utilizando métodos compartidos y métodos de instancia (a la inversa, no se puede provocar un suceso regular a partir de un método compartido).

349. ¿Para qué se utiliza AddHandler?

Permite decidir en tiempo de ejecución que rutina deberá dar servicio a un determinado suceso. AddHandler acepta 2 argumentos: el suceso que voy a redirigir a un procedimiento de la aplicación (en el formato objeto.nombresuceso) y la dirección de la rutina que controla el suceso (en el formato AddressOf nombrerutina).

350. ¿se pueden atrapar sucesos desde matrices? ¿Cómo?

Si, utilizando AddHandler.

351. ¿Qué son los miembros compartidos?

Contestado en las preguntas siguientes

352. ¿Qué característica posee un campo compartido?

Los campos compartidos son variables que pueden ser accedidos (es decir, compartidos) por todas las instancias de una clase determinada. Hay que ponerle la palabra Shared adelante. Ej: Shared Nombre as String. Los campos compartidos también resultan de utilidad cuando todas las instancias de una clase compiten entre sí por un número limitado de recursos.

353. ¿Qué característica posee un método compartido?

Se utiliza la palabra Shared para marcar un método como estático, lo que permitirá llamar al método sin necesidad de generar una instancia del objeto de dicha clase.

354. ¿Qué característica posee un constructor compartido?

Si se define un método Shared Sub New sin parámetros en una clase, justo antes de que se genere la primera instancia de esta clase se llamará automáticamente a este procedimiento. Normalmente se utilizará este tipo de constructor compartido para iniciar correctamente los campos compartidos.

El procedimiento Shared Sub New se ejecuta antes que el procedimiento Sub New para el primer objeto instanciado de la clase. Los constructores compartidos son implícitamente Private y no se pueden declarar utilizando el calificador de ámbito Public o Friend. Los constructores compartidos son el único lugar en el que se puede asignar un campo Shared ReadOnly.

355. ¿Qué característica posee un suceso compartido?

Los sucesos compartidos se definen de la misma forma en que se definen campos, métodos y propiedades compartidas. Se puede provocar sucesos compartidos utilizando métodos compartidos y métodos de instancia (a la inversa, no se puede provocar un suceso regular a partir de un método compartido).

356. ¿Qué son y para que se pueden utilizar las clases anidadas?

Las clases anidadas tienen un gran número de propósitos. En primer lugar, suelen ser útiles para organizar las clases en grupos de clases relacionadas y para crear espacios de nombre que ayuden a resolver las ambigüedades de nombres. Otro empleo frecuente de las clases anidadas es encerrar una o más clases auxiliares dentro de la clase que las utiliza y evitar hacerlas visibles a otras partes de la aplicación. En este caso, la clase más interna deberá estar marcada con el calificador de ámbito Private.

357. ¿Qué ámbitos existen?

*Público
*Privado
*Friend
*Protected
*Protected Friend

358. ¿Qué característica posee el ámbito público?

Hace que una clase, o cualquiera de sus miembros, sea visible fuera del ensamblado actual si el proyecto es un proyecto de biblioteca.

359. ¿Qué característica posee el ámbito privado?

Convierte en privada a una clase y utilizable únicamente dentro de su contenedor. Este contenedor suele ser normalmente la aplicación actual salvo en el caso de las clases anidadas (una clase anidada privada sólo se puede utilizar dentro de su clase contenedora). Un miembro privado es utilizable únicamente dentro de la clase en la que se ha definido y esto incluye cualquier clase anidada en el mismo contenedor.

360. ¿Qué característica posee el ámbito friend?

Hace que una clase (o cualquiera de sus miembros) sea visible en el ensamblado actual. Éste es el ámbito predeterminado para las clases.

361. ¿Qué característica posee el ámbito protected?

Hace que un miembro o una clase anidada sea visible dentro de la clase actual así como en todas las clases derivadas de la clase actual. Los miembros protected son miembros privados que también son heredados por las clases derivadas.

362. ¿Qué característica posee el ámbito protected friend?

Combina las características de las palabras clave Friend y Protected y, por tanto, define un miembro o clase anidada que es visible en todo el ensamblado y en todas las clases heredadas.

UNIDAD 6

363. ¿Qué es la herencia?

Es la capacidad de derivar una nueva clase (la clase derivada o heredada) de una clase más sencilla (la clase base). La clase derivada hereda todos los campos, propiedades y métodos para reemplazarlas. También podrá agregar nuevos campos, propiedades y métodos a las clases heredadas. La herencia es especialmente eficaz para representar una relación del tipo “es un” entre dos clases. La herencia es una forma eficaz de reutilizar código en una clase.

364. ¿Qué cosas se pueden heredar?

La clase derivada hereda todas las propiedades, métodos y campos Public y Friend de la clase base.

365. ¿Cómo y para qué se puede aprovechar en la práctica el polimorfismo?

Para la reutilización de código y mayor eficiencia y legibilidad del mismo. En la clase base colocar antes de la firma del método Overridable. En la clase derivada colocar Overrides

366. ¿Cómo y para qué se utiliza la clase derived?

Se puede utilizar sin tener que conocer que deriva de otra clase. Sin embargo, conocer la relación de herencia entre las dos clases ayuda a escribir código más flexible.

367. ¿Qué representa la clase me?

Representa el contexto actual de la clase en ejecución.

368. ¿Qué clase representa a la clase base?

MyBase

369. ¿Para qué utilizaría MyBase?

Resulta de utilidad cuando se desea hacer referencia a un campo, propiedad o método del objeto base.

370. ¿Qué representa la clase MyClass?

Se utiliza para asegurarse de que lo métodos existentes en una clase base siempre utilizarán las propiedades y métodos contenidos en dicha clase base (en oposición a la versión sobrescrita de la clase heredada).

371. ¿Qué diferencia existe entre MyBase y MyClass?

La diferencia es que MyBase representa a la clase de la que heredo, en cambio MyClass usa el campo de la clase donde defino el MyClass, ignora el contexto de ejecución.

372. ¿Para qué se usa una clase abstracta?

Se caracteriza porque sólo puede ser heredada pero no instanciada.

373. ¿Para qué se usa una clase sellada?

Se puede instanciar pero no heredar. Se antepone la palabra clave NotInheritable

374. ¿Qué es la sobreescritura?

La sobre-escritura, se aplica a los métodos y está directamente relacionada a la herencia y se refiere a la re-definición de los métodos de la clase base en las subclases.

375. ¿Qué elementos se pueden sobreescribir?

Los métodos.

376. ¿A qué se denomina sombreado de métodos?

Si dos elementos de programación comparten el mismo nombre, uno de ellos puede ocultar o sombrear al otro. En esta situación, el elemento sombreado no está disponible como referencia; en vez de esto, cuando el código utiliza el nombre del elemento, el compilador de Visual Basic resuelve en favor del elemento que sombrea.

377. ¿Qué característica peculiar posee el sombrado vs la sobre-escritura?

Que sobre escribe y oculta los otros métodos de la sobrecarga. Un miembro de la clase derivada, marcado con la palabra clave Shadows, oculta todos los miembros de la clase base que tengan el mismo nombre, con independencia de sus firmas. Un miembro de la clase derivada, marcado con la palabra clave Overloads, sombreará únicamente al miembro contenido en la clase base que tenga el mismo nombre y firma de argumentos.

UNIDAD 7

378. ¿Para qué se utilizan las interfaces?

Para tipar. Una interfaz es el conjunto de propiedades y métodos que expone una clase. Una interfaz define únicamente la firma de tales propiedades y métodos (nombre del miembro, número y tipo de cada parámetro y tipo del valor retornado), mientras que la clase implementará dicha interfaz proporcionando el código necesario para el correcto funcionamiento de dichas propiedades y métodos. El código asociado a cada propiedad o método puede ser distinto de una clase a otra, aunque se conservará la semántica de cada método. El hecho de que cada clase pueda implementar la misma propiedad o método de una forma distinta es la base del polimorfismo. El ámbito predeterminado de las interfaces es Friend.

379. ¿Cómo se implementa una interfaz?

Public Interface xx … End Interface

380. ¿Se pueden heredar las interfaces?

Si, una interfaz puede heredar de otra interfaz. Una interfaz heredada contendrá todos los miembros definidos en ella además de todos los miembros definidos en la interfaz base.

381. ¿Se puede implementar un tipo de polimorfismo peculiar por medio de interfaces?

Si.

382. ¿Para qué se utiliza la interfaz IComparable?

Para ordenar únicamente por un solo criterio de ordenamiento. Los objetos sólo pueden compararse de una forma.

383. ¿Para qué se utiliza la interfaz IComparer?

La mayoría de los objetos del mundo real se pueden comparar y ordenar atendiendo al contenido de diferentes campos o combinaciones de campos, en este caso se podrá utilizar una variedad del método Array.Sort que acepta una interfaz IComparer como segundo argumento. Cualquier clase que pueda ser ordenada atendiendo a diferentes combinaciones de campos, puede exponer dos o más clases anidadas que implementen la interfaz IComparer, una clase para cada posible método de ordenación.

384. ¿Para qué se utiliza la interfaz IClonable?

Admite la clonación, que crea una nueva instancia de una clase con el mismo valor que una instancia existente. El Framework .NET provee a todos los objetos un método llamado MembershipClone() que crea una copia simple del objeto. Una copia simple es una copia de los miembros de la clase sin realizar una copia de los objetos a los que estos miembros apuntan. ¿Qué pasa entonces cuando queremos copiar objetos de estructuras complejas que tienen atributos que contienen objetos, que contienen atributos que contienen objetos, y así sucesivamente? Se puede implementar la interfaz IClonable que soporta el método Clone(), para soportar explícitamente una copia profunda de la clase. De todas formas, esto requiere de un trabajo de creación de objetos y asignación manual de valores que debe hacerse por líneas de código. Esta tarea podría resultar muy ardua si la estructura del objeto a clonar es muy grande, y requiere un mantenimiento extra si la estructura de la clase se modifica.

385. ¿Para qué se utiliza la interfaz IEnumerable?

Enumera elementos de una clase desde afuera de la misma, entregando una lista de elementos y no la colección completa. Obliga a implementar getEnumerator() que devuelve un Enumerador. Recupera todos los elementos de un enumerador. Esta interfaz deberá devolver un objeto que permita el empleo de la interfaz Ienumerator.

386. ¿Para qué se utiliza la interfaz IEnumerator?

Para crear un numerador personalizado. Posee los siguientes métodos

- Funcion MoveNext: Este método se llama en cada una de las iteraciones ForEach y debe devolver True si existe un nuevo valor o False si no existen más elementos.
- Propiedad de solo lectura Current: Devuelve el valor actual.
- Procedimiento Reset: Este método redefine el puntero interno para que el siguiente valor que se devuelva sea el primero de una nueva serie.

Estas dos interfaces funcionan conjuntamente para proporcionar compatibilidad For Each para que las clases se comporten como una clase colección ante sus clientes.

387. ¿Qué es un delegado?

Es similar a un puntero a una función de C en el sentido que permite llamar a un procedimiento utilizando un puntero que apunte al propio procedimiento.

388. ¿A qué elementos se les puede delegar?

* Cada delegado sólo puede invocar a aquellos procedimientos que tengan una determinada firma de argumentos que se deberá especificar en la declaración del delegado.

*Un delegado puede apuntar a un procedimiento estático o a un procedimiento de instancia.

389. ¿Qué se puede delegar?

* Eventos
*Funciones

390. ¿Cómo construiría un delegado?

Dim deleg as SubunArg
deleg= New SubUnArg (AdressOf MostrarMsg)
Delegate Sub SubUnArgr (ByVal msg as String)

391. ¿Cómo implementaría un procedimiento de devolución de llamadas?

Un procedimiento de devolución de llamadas es un procedimiento contenido en el programa al que llama otra rutina cuando tenga algo que notificar.

392. ¿Para qué sirve la multidifusión de delegados?

Para enviar una llamada a más de un procedimiento.

393. ¿Qué es un atributo?

Los atributos son la forma de indicar dentro del código fuente que ciertos métodos o clases tienen un comportamiento especial definido mediante dichos atributos, amén de poder incluir toda una serie de metadatos dentro del código ejecutable para que sirva a nuestros propósitos o a los del entorno de ejecución. Cuando se compila el código para el motor en tiempo de ejecución, se convierte a lenguaje intermedio de Microsoft (MSIL) y se coloca en un archivo ejecutable portable (PE) junto con los metadatos generados por el compilador. Los atributos permiten colocar información descriptiva adicional en metadatos que se pueden extraer utilizando servicios de reflexión en

tiempo de ejecución. El compilador crea atributos cuando se declaran instancias de clases especiales que se derivan de System.Attribute.

.NET Framework utiliza atributos por varias razones y para tratar una serie de problemas. Los atributos describen cómo se serializan datos, se especifican características que se utilizan para exigir seguridad y se limitan optimizaciones mediante el compilador Just-in-time (JIT) para facilitar la depuración del código. También pueden grabar el nombre de un archivo o el autor del código, o controlar la visibilidad de controles y miembros durante el desarrollo de formularios.

394. ¿Cómo es la sintaxis de un atributo?

```
<System.ComponentModel.DescriptionAttribute("Una persona")> _
Class Persona
' . . .
End Class
```

395. ¿Para qué se utiliza el atributo StructLayout?

Se suele aplicar se pasa una estructura a un código no administrado en una DLL. Estos atributos se pueden aplicar también a las clases y a los elementos de clase, incluso aunque este último empleo suele ser menos frecuente.

396. ¿Para qué se utiliza el atributo FieldOffset?

Se utiliza para implementar una unión (una estructura con dos o más elementos que se superponen en memoria)

397. ¿Para qué se utiliza el atributo Conditional?

Se utiliza para indicar a los compiladores que se debería omitir una llamada al método o atributo a menos que se defina un símbolo de compilación condicional especificado. Visual Basic siempre ha ofrecido una forma de incluir o de excluir trozos de código en el código compilado, para ello se utilizan las directivas #If.

398. ¿Para qué se utiliza el atributo Obsolete?

Se utiliza para marcar los elementos del programa que ya no se utilizan. Obsolete es aplicable a todos los elementos de programa con excepción de ensamblados, módulos, parámetros o valores devueltos. Al marcar un elemento como obsoleto, se informa a los usuarios de que el elemento se quitará en versiones futuras del producto.

399. ¿Para qué se utiliza el atributo DebuggerStepThrough?

El atributo DebuggerStepThrough se utiliza para marcar una función que el depurador de Visual Basic .NET deba ignorar porque deba ejecutarse como un todo o, simplemente, porque ya ha sido probada y depurada.

UNIDAD 8

1. ¿Qué características posee un esquema cliente - servidor?

Se tiene una máquina cliente, que requiere un servicio de una máquina servidor, y éste realiza la función para la que está programado (nótese que no tienen que tratarse de máquinas diferentes; es decir, una computadora por sí sola puede ser ambos cliente y servidor dependiendo del software de configuración). Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma

2. ¿Qué significa pasar información batch?

Implica procesar varias transacciones al mismo tiempo, y no se dispone inmediatamente de los resultados del resto de transacciones cuando comienza cada una de ellas para un mejor funcionamiento de un sistema.

3. ¿Qué significa pasar información on line?

Online es una palabra inglesa que significa “en línea”. El concepto se utiliza para nombrar a algo que está conectado o a alguien que está haciendo uso de una red (generalmente, Internet). Se dice que la información está online o en línea, cuando se encuentra disponible a través de Internet.

4. ¿Qué es un protocolo?

Es un conjunto de reglas usadas por computadoras para comunicarse unas con otras a través de una red por medio de intercambio de mensajes. Éste es una regla o estándar que controla o permite la comunicación en su forma más simple, puede ser definido como las reglas que dominan la sintaxis, semántica y sincronización de la comunicación. Los protocolos pueden ser implementados por hardware, software, o una combinación de ambos. A su más bajo nivel, éste define el comportamiento de una conexión de hardware.

5. ¿Qué protocolo usa una red de área local?

TCP/IP

6. ¿Qué protocolo usa internet?

IP

7. ¿Qué hace el protocolo IP?

IP es el protocolo encargado del transporte de paquetes desde el origen hasta el destino en una comunicación. Es un protocolo que no garantiza la fiabilidad aunque trata de hacer todo lo posible para que los paquetes lleguen al destino. IP se basa en el encaminamiento que se produce entre dispositivos de la capa 3 (OSI) y en los identificadores únicos (direcciones IP) que asigna a cada dispositivo para que pueda comunicarse. Cada dispositivo de capa 3 realiza el proceso de encaminamiento en base a su tabla de rutas, que le indica el camino más adecuado para llegar a cada destino.

8. ¿Qué ventajas tiene distribuir procesos?

Los sistemas distribuidos están basados en las ideas básicas de transparencia, eficiencia, flexibilidad, escalabilidad y fiabilidad. Por lo tanto los sistemas distribuidos han de cumplir en su diseño el compromiso de que todos los puntos anteriores sean solucionados de manera aceptable.

9. ¿Qué ventajas tiene distribuir almacenamientos?

Se conoce como sistema de almacenamiento distribuido a todo aquel que permite almacenar ficheros online. La principal característica es poder guardar archivos (documentos, imágenes, vídeos...) en la red. Una de las cosas que caracterizan al almacenamiento distribuido es el gran rango de aplicaciones que tiene. Las tres más importantes son:

*Copias de seguridad de los archivos.
*Compartir archivos en red.

10. ¿Qué es un socket?

Un Socket es una relación entre un puerto de un equipo y el puerto de otro equipo. Los puertos son básicamente, una entrada/salida de información. Estos se encuentran identificados por un número entero y muchos se encuentran reservados para determinadas tareas, por ejemplo: el puerto 80 es para un servidor web.

11. ¿Qué característica posee un socket sincrónico?

Quien envía permanece bloqueado esperando a que llegue una respuesta del receptor antes de realizar cualquier otro ejercicio.

12. ¿Qué característica posee un socket asincrónico?

Quien envía continúa con su ejecución inmediatamente después de enviar el mensaje al receptor.

13. ¿Qué objeto se puede utilizar para construir un navegador?

14. ¿Para qué se utilizan los puertos de la pc?

Sirven para comunicar nuestro ordenador con los periféricos u otros ordenadores. Se trata en definitiva de dispositivos I/O (Input/Output, o Entrada/Salida).

15. ¿Qué puertos posee una PC?

Paralelo – Serie – USB- FireWire

16. ¿Cómo funciona el puerto paralelo?

En un esquema de transmisión de datos en paralelo un dispositivo envía datos a otro a una tasa de n número de bits a través de n número de cables a un tiempo. Sería fácil pensar que un sistema en paralelo es n veces más rápido que un sistema en serie, sin embargo esto no se cumple, básicamente el impedimento principal es el tipo de cable que se utiliza para interconectar los equipos.

17. ¿Cómo funciona el puerto serie?

En un esquema de transmisión de datos en serie un dispositivo envía datos a otro a razón de un bit a la vez a través de un cable.

18. ¿Cómo funciona el puerto USB?

Un buen punto de partida para abordar este tema es el cableado del bus. Cada cable USB contiene, a su vez, 4 cables en su interior. Dos de ellos están dedicados a la alimentación (5 voltios) y la referencia de tensión (masa). La corriente máxima que el bus puede proporcionar es de 500 mA a 5 voltios de tensión.

Los dos cables restantes forman un par trenzado, que transporta la información intercambiada entre dispositivos, en formato serie. Tras su encendido, el dispositivo anfitrión -el PC- se comunica con todos los dispositivos conectados al bus USB, asignando una dirección única a cada uno de ellos (este proceso recibe el nombre de “enumeración”).

Además, la PC consulta qué modo de transferencia desea emplear cada dispositivo: por interrupciones, por bloques o en modo isócrono.

La transferencia por interrupciones la emplean los dispositivos más lentos, que envían información con poca frecuencia (por ejemplo teclados, ratones, etc.). La transferencia por bloques se utiliza con dispositivos que mueven grandes paquetes de información en cada transferencia. Un ejemplo son las impresoras. Finalmente, la transferencia isócrona se emplea cuando se requiere un flujo de datos constante y en tiempo real, sin aplicar detección ni corrección de errores. Un ejemplo es el envío de sonido a altavoces USB. Como se puede intuir, el modo isócrono consume un ancho de banda significativo. Por ello el PC impide este tipo de transferencia cuando el ancho de banda consumido supera el 90% del ancho de banda disponible.

Para la temporización, el bus USB divide el ancho de banda en porciones, controladas por el PC. Cada porción mueve 1.500 bytes, y se inicia cada milisegundo. Ante todo, el PC asigna ancho de banda a los dispositivos que emplean transferencias isócronas y por interrupciones, garantizando el ancho de banda necesario. Las transferencias por bloques emplean el espacio restante, quedando en última prioridad.

19. ¿Qué es la domótica?

La domótica es la integración de tecnología en el diseño inteligente o automatizado de un recinto (Casa, Apartamento, Casas campestres, fincas, lugar de trabajo etc…) con funciones de información, entretenimiento, gestión energética, seguridad y búsqueda de soluciones a la medida y aplicaciones según sus necesidades. Entonces, se trata de una ciencia que estudia la aplicación de la informática y las comunicaciones al hogar, con el fin de conseguir una -casa inteligente-. La domótica pretende, por ejemplo, que las luces, calefacción, etc., se regulen automáticamente en función de las condiciones exteriores, consiguiendo de paso un considerable ahorro energético.