PROGRAMACIÓN ORIENTADA A OBJETOS

Profesor: Christian Parkinson

2018

FACULTAD DE TECNOLOGÍA INFORMÁTICA

Alumno: Patricio Fernandez

UNIVERSIDAD ABIERTA INTERAMERICANA

INDICE

Contenido

GUÍA DE TRABAJOS REVISIÓN CONCEPTUAL 3
UNIDAD I 3
UNIDAD II 29
UNIDAD III 34
UNIDAD V 43
GUÍA DE TRABAJOS PRÁCTICOS 49
UNIDAD II 50
UNIDAD III 51
UNIDAD IV 51
UNIDAD V 51
UNIDAD VI 52
GUÍA DE ABORDAJE BIBLIOGRÁFICO 52
UNIDAD I 52
UNIDAD II 53
UNIDAD III 53
UNIDAD IV 53
UNIDAD V 53
UNIDAD VI 54

GUÍA DE TRABAJOS REVISIÓN CONCEPTUAL

UNIDAD I

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

- La complejidad del dominio del problema: Los usuarios normalmente tienen dificultad al expresar con precisión sus necesidades de una forma en la cual los desarrolladores puedan comprender. Los usuarios y desarrolladores tienen perspectivas diferentes sobre la naturaleza del problema y realizan distintas suposiciones sobre la naturaleza de la solución. Además, otra complicación es que los requisitos cambian frecuentemente durante su desarrollo, por lo que el sistema evoluciona.

- La dificultad de gestionar el proceso de desarrollo: Se intenta que el sistema en desarrollo sea lo más comprensible posible, pero a veces el sistema es muy grande por lo que puede contener ciento de miles de líneas de código dividas en módulos. De esta manera, es complicado comprender un sistema de gran envergadura, por lo cual se requiere un gran equipo de desarrollo que también puede estar ubicado en diferentes partes del mundo, lo que también genera complejidad.

- La flexibilidad que se puede alcanzar a través del Software: 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 sí 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: 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. Sin embargo, sigue dándose el hecho de que las transiciones de fase entre estados discretos no pueden modelarse con funciones continuas. Todos los eventos externos a un sistema de software 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. En las peores circunstancias, 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?

- La complejidad toma forma de jerarquía. Sistemas complejos que se componen de subsistemas relacionados que a su vez tienen sus propios subsistemas. Por lo cual, se ven los sistemas por partes. Relaciones entre las partes.

- Componentes primitivos. Quedan a la clasificación del observador.
    - Alta frecuencia de componentes: Involucra la estructura interna de estos.
    - Baja Frecuencia: Involucra la interacción entre los componentes.
    - Esto posibilita el estudio de cada parte de forma aislada.
        - ✓ Sistemas jerárquicos descomponibles: Se pueden dividir en partes identificables
        - ✓ Sistemas jerárquicos casi descomponibles: Partes no completamente independientes.

- Sistemas con economía de expresión. Los sistemas complejos tienen patrones comunes, los cuales pueden conllevar a la reutilización de componentes pequeños.
- Sistemas complejos tienden a evolucionar durante el tiempo. A medida que los sistemas evolucionan, objetos que antes se consideraban complejos ahora se consideran primitivos.

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

Las dos jerarquías más importantes que consideramos en la orientación a objetos para sistemas complejos son:

- **Estructura de clases**: Jerarquía estructural o "parte-de" (part of).
- **Estructura de objetos:** Jerarquía de tipos "es-un" (is a).

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 vencerás. 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:
- Orientada a objetos.
- Algorítmica.

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

La división algorítmica enfatiza el orden de los eventos, y la visión orientada a objetos resalta los agentes que causan acciones, o bien, son sujetos de estas acciones. No se puede construir un sistema complejo de ambas formas a la vez ya que son vistas de forma perpendicular.

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

La abstracción denota las características esenciales de un objeto que lo diferencian de los demás y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador.

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

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

Lo considero un desarrollo evolutivo, ya que no destruye los avances del pasado; adapta los nuevos requerimientos al diseño actual.

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

- Modelo dinámico.
- Modelo estático.
- Modelo Lógico (estructura de clases y de objetos).
- Modelo físico (Arquitectura de módulos y de procesos).

11. ¿Qué es la programación orientada a objetos?

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.

Un lenguaje es orientado a objetos si:

- Soporta objetos, abstracciones de datos con una interfaz de operaciones con nombre y estado local oculto.
- Los objetos tienen un tipo asociado (clase).

- Los tipos [Clase] pueden heredar atributos de los supertipos [superclases] como soportar herencia significa que es posible expresar relaciones “es un” entre tipos.

12. ¿Qué es el diseño orientado a objetos?

Es 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 estático y dinámico del sistema que se diseña.

- Modelo lógico: Estructura de clases y objetos.
- Modelo Físico: Arquitectura de módulos y procesos.

13. ¿Qué es el análisis orientado a objetos?

El análisis orientado a objetos es un método de análisis que examina los requisitos desde la perspectiva de clases y objetos que se encuentran en el vocabulario del dominio del problema.

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

Los elementos fundamentales en el modelo de objetos son:

- Abstracción
- Encapsulamiento.
- Modularidad.
- Jerarquía.

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

Los modelos secundarios del modelo de objetos son:

- Concurrencia
- Persistencia
- Tipos (Tipificación).

16. Explique el significado de la abstracción.

La abstracción es una vía fundamental por la que los humanos combatimos la complejidad. Denota 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. Tienen propiedades estáticas y dinámicas.

17. Explique el significado del encapsulamiento.

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

18. Explique el significado del modularidad.

Modularidad es una propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados. Ofrece una vía para agrupar abstracciones relacionadas lógicamente.

19. Explique el significado de la jerarquía.

La jerarquía es una clasificación u ordenación de abstracciones. Las jerarquías más importantes de un sistema complejo son su estructura de clases (la jerarquía "de clases"), y su estructura de objetos (la jerarquía "de partes).

20. Explique el significado de la tipificación.

La tipificación o 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 solo de formas muy restringidas.

21. Explique el significado de la concurrencia.

La concurrencia es manejar muchos eventos diferentes simultáneamente. Un sistema que implique concurrencia puede tener muchos hilos de control. Esta se centra en la abstracción de procesos y sincronización. La concurrencia es la propiedad que distingue un objeto que está activo de uno que no lo está.

22. Explique el significado de la persistencia.

La propiedad es 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 (la posición del objeto varia con respecto al espacio de direcciones en el que fue creado)

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?

Las características esenciales se denotan por medio de la abstracción.

24. ¿A qué denominamos un objeto cliente?

Un objeto cliente es el que puede operar sobre otros objetos, pero nunca se opera sobre él por parte de otros objetos. Objeto activo y actor son equivalentes en algunos casos

25. ¿A qué denominamos un objeto servidor?

Un objeto servidor nunca opera sobre otros objetos; solo otros objetos operan sobre él.

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

Según Meyer 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 comportamiento 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 que cada objeto cumpla con su responsabilidad, para lograr un fin común a través de las Precondiciones y las Postcondiciones.

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?

Todos los métodos y subprogramas libres asociados a un objeto concretado forman un protocolo. El protocolo define así la envoltura del comportamiento admisible en un objeto.

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?

Se lo denomina protocolo al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con todas las formas de invocación u órdenes que admite.

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 no se mantiene, se rompe el contrato asociado con una abstracción.

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

Se deben definir pre-condiciones y póst-condiciones.

32. ¿Qué es una pre-condición?

Es una condición que debe cumplirse cuando la operación inicia su ejecución. En caso de no cumplirse, se podrá de forma legítima, rehusar su ejecución y posiblemente disparar alguna excepción, según sea el caso.

33. ¿Qué es una post-condición?

Es una condición que debe ser fiable cuando la operación termina su ejecución. Sí no lo es, entonces la implementación de la operación tiene defectos y deberán ser corregidos cuanto antes.

Los invariantes de clase, junto con las pre-condiciones y post-condiciones de las operaciones forman un marco de trabajo 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?

La excepción es una condición que no se ha logrado satisfacer alguna invariante y se genera cuando se produce un acontecimiento circunstancial que impide el normal funcionamiento del programa, llamado así comportamiento “No Deseado”

35. ¿A qué se denomina mensaje?

Es la operación que se realiza sobre un objeto, básicamente es una orden que se envía a un objeto para indicarle que realice alguna acción, llamado “pase de mensajes”. Logrando así que 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 el encapsulamiento se centra en la implementación que da lugar a ese 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.

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

Se denomina Interfaz y proporciona su visión externa, ocultando secretos y enfatizando la abstracción.

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

Se denomina Implementación y engloba los secretos de su comportamiento, se compone principalmente por todas las operaciones definidas en la interfaz de la misma.

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.

El concepto mencionado apunta a prevenir el mal uso de la información, se evita el uso inapropiado de los detalles de implementación. Se trata de tener encapsulada u oculta esa información en el estado interno del objeto y no visiblemente. Nos interesa saber qué hace, pero no cómo lo hace.

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

Los dos aspectos son:

- La cohesión.
- Acoplamiento.

41. ¿Para qué se utiliza la jerarquía?

La jerarquía se utiliza para la clasificación y el orden de las abstracciones.

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

La denominamos tipo y no pueden intercambiarse o solo lo pueden hacer de forma muy restringida.

43. ¿Las clases implementan tipos?
Si

44. ¿Los tipos implementan clases?
No.

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

Los denominados lenguajes tipados, o lenguajes de tipos

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

Lo denominados no tipados o débilmente tipados.

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

Asignación estática de tipos, o también conocida como ligadura estática temprana, se refiere al momento en que los nombres se ligan a sus tipos. Se fijan los tipos de las variables y expresiones en tiempo de compilación.

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

La ligadura dinámica o bien llamada ligadura 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, al ser la comprobación estricta de tipos y la ligadura de conceptos independientes, un lenguaje puede tener comprobación estricta de tipos pero soportar enlace dinámico, o no tener tipos y admitir la ligadura dinámica.

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

Se denomina Concurrencia y justamente es un sistema automatizado para manejar muchos eventos diferentes simultáneamente. Puede tener muchos hilos de control. Esta se centra en la abstracción de procesos y sincronización. La concurrencia distingue un objeto activo de que uno inactivo.

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.

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

Un proceso ligero o liviano suele existir dentro de un solo proceso del sistema operativo en compañía de otros procesos ligeros y que comparten el mismo espacio de direcciones. La comunicación entre sistemas ligeros suelen ser menos costosas e involucran datos compartidos.

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

Si, esa es su función.

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

Se denomina Persistencia.

55. ¿Qué cosas se persisten?

Persisten entre los objetos de:

- Los resultados transitorios de evaluación de expresiones.
- En las variables locales en la activación de procedimientos.
- En las variables propias, cuya duración difiere de su hábito (ALGOL 60)
- Datos que existen entre ejecuciones de programas.
- Datos que existen entre varias versiones de un programa
- Datos que sobreviven a un programa.

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, o visible, que se puede tocar y comprender intelectualmente. Algo que redirige una acción o pensamiento.

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?

Algunos objetos pueden tener límites conceptuales precisos, pero aun así pueden representar eventos o procesos intangibles. Algunos objetos pueden ser tangibles y aun así tener fronteras físicas difusas. O sea, pueden poseer ambas.

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

Las tres cosas que debe tener un objeto son:

- Estado.
- Comportamiento.
- Identidad.

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

Como sinónimo de objeto se puede utilizar la palabra Instancia.

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

Como sinónimo se utiliza Objeto.

62. ¿Cómo definiría el estado de un objeto?

El estado de un objeto abarca todas las propiedades del mismo (normalmente estáticas) más los valores actuales de cada una de esas propiedades (normalmente dinámicas) todo objeto tiene un estado que implica una cierta cantidad del espacio, ya sea en el mundo físico o en la memoria de la computadora.

63. ¿A qué definimos propiedad de un objeto?

Una propiedad es una característica inherente o distintiva, un rasgo o cualidad que contribuye a hacer que in objeto sea ese objeto y no otro.

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

Todas las propiedades tienen un valor, y este valor puede ser una mera cantidad, o puede denotar a otro objeto. Así se distingue entre un objeto y los valores simples:
Los valores simples son:

- Intemporales.
- Iinmutables.
- No instanciadas.

Mientras que los objetos existen en el tiempo, y puede ser:

- Modificables.
- Tienen estado.
- Son instanciados y pueden crearse, destruirse y compartirse

65. ¿Cómo definiría el comportamiento de un objeto?

El comportamiento es como actúa y reacciona un objeto, en términos de sus cambios de estado y paso de mensajes. Representa su actividad visible y comprobable exteriormente.

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 el efecto lateral de modificar el estado de objeto.

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

Sí, porque el estado de un objeto representa los resultados acumulados de su comportamiento.

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

Si, se puede afirmar.

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

Una operación denota un servicio que una clase ofrece a sus clientes, es decir que las operaciones que los clientes pueden realizar sobre un objeto suelen declararse como métodos.

70. ¿Cuáles son las tres operaciones más comunes?

Las operaciones más comunes son:

- Modificador.
- Selector.
- Iterador.

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

Las dos operaciones más habituales que se utilizan son:

- Constructor.
- Destructor.

72. ¿Qué tipo de operación es el modificador?

Es un tipo de operación que altera el estado de un objeto.

73. ¿Qué tipo de operación es el selector?

Es un tipo de operación que accede al estado de un objeto, pero no altera su estado.

74. ¿Qué tipo de operación es el iterador?

Es 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?

Es una operación que crea un objeto y/o analiza su estado.

76. ¿Qué tipo de operación es el destructor?

Es 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. Cfifi)?

Los lenguajes como C++ permite al desarrollador escribir operaciones como subprogramas libres, se las llaman funciones No Miembros. Todos los subprogramas libres son procedimientos o funciones que sirven como operaciones no primitivas sobre un objeto de la misma o distinta clase. Los subprogramas están típicamente agrupados según las clases sobre las que se ha construido (utilidades de clase)

78. ¿Todos los métodos son operaciones?

Si, por supuesto. Todos los métodos son operaciones.

79. ¿Todas las operaciones son métodos?

No, todas las operaciones no son métodos. Algunas se pueden denominar 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?

Sí, es conveniente. Para la mayoría de las abstracciones no trivales es útil dividir este protocolo en grupos lógicos de comportamiento. Estas colecciones, que constituyen una partición del

espacio de comportamiento de un objeto, denotan papeles que un objeto puede desempeñar. Un papel es una máscara que se pone un objeto y por tanto define un contrato entre una abstracción y sus clientes

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

Las denominamos 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?

Sí, es relevante el orden en que se invocan las operaciones de un objeto, como por ejemplo una máquina expendedora de dulces, se selecciona el producto, la maquina registra si hay stock, y el mismo expende el producto luego.

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

Siguiendo con el ejemplo de la respuesta del punto 84, cada objeto es como una máquina independiente. Para algunos objetos, éste orden 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 equivalentes.

85. ¿Qué es la identidad de un objeto?

La identidad es aquella propiedad de un objeto que lo distingue de todos los demás objetos.

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

Son dos variables que apuntan a 2 instancias, pero con el mismo estado u objetivo.

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

Significa que 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?

Ambas van a apuntar al mismo objeto en memoria y pueden tener el mismo estado o no.

89. ¿Qué significa realizar una clonación superficial?

Significa que copia el objeto y comparte su estado.

90. ¿Qué significa realizar una clonación profunda?

Significa que copia el objeto, así como su estado y así sucesivamente con los demás objetos que componen el objeto a clonar.

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

El ciclo o espacio de vida de un objeto se extiende desde el momento en que se crea por primera vez (y así consume espacio por primera vez) hasta que ese espacio se recupera. Al mismo hay que declararle o asignarle memoria de forma dinámica.

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

Se debe usar la clase garbage collector: ej: GC.Collect();

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

Los tipos de relaciones que existen entre los objetos son:

- Enlaces: El concepto deriva de Rumbaugh, que lo define como una conexión física o conceptual entre objetos. Un enlace denota la asociación específica por la cual un objeto utiliza los servicios de otro objeto.
- Agregación: Denota una jerarquía todo/parte, con la capacidad de ir desde el todo hasta sus partes. La agregación es un tipo de especializado de asociación.

94. ¿Cómo podemos definir al enlace 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, es decir un cliente, y utiliza los servicios de otro objeto, es decir un 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 puede ser unidireccional, aunque, también puede ser bidireccional.

96. ¿Qué es un mensaje unidireccional?

Un mensaje unidireccionales es cuando un objeto activo invoca a un objeto o agente que actue como pasivo.

97. ¿Qué es un mensaje bidireccional?

Un mensaje bidireccional es cuando un 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?

Inicia el paso de mensajes lo inicia el cliente u objeto activo.

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

Un objeto puede desempeñar tres papeles o roles:

- Actor.
- Servidor.
- Agente.

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

Un objeto puede actuar como Actor sobre otros objetos pero nunca opera sobre el por parte de otros objetos, en algunos contextos, los términos objeto activo y actor son equivalentes.

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

Es un objeto que nunca opera sobre otros objetos, solo otros objetos operan sobre el.

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

Significa que un objeto puede operar sobre otros objetos y otros objetos pueden operar sobre él. Puede un agente normalmente operar para realizar algún trabajo en nombre de un actor u agente.

103. Dados dos objetos A y fí, si A le puede enviar un mensaje a fí, estando ambos Relacionados por enlace, decimos que fí respecto de A están:

Visible.

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

Las cuatro visibilidades que puede poseer un objeto servidor son:

- Objeto servidor global para el cliente.
- Objeto servidor como parámetro para alguna operación de cliente.
- Objeto servidor como parte de un objeto cliente.
- Objeto servidor como 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 están:

Sincronizados.

106. ¿Cuáles son las posibles formas de sincronización?

Las posibles formas de sincronización pueden ser:

- Secuenciales:
- Vigilado.
- Síncrono.

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

La semántica del objeto pasivo está garantizada solo en presencia de un único objeto activo simultáneamente.

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

La semántica del objeto pasivo está garantizada en presencia de múltiples hilos de control, pero los clientes activos deben colaborar para lograr la exclusión mutua.

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

La semántica del objeto pasivo está garantizada en presencia de múltiples hilos de control y el servidor garantiza exclusión muta.

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?

Es meramente jerárquica.

112. ¿Qué tipo de jerarquía denota la agregación?

La agregación denota una jerarquía todo/parte, con la capacidad de ir desde el todo hasta sus partes.

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

Recibe también el nombre de agregado o contenedor.

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

Si, por supuesto.

115. ¿Qué tipos de agregación existen?

Existen dos tipos de agregaciones:

- Física: Cuando un objeto no existe sin el otro. Sus ciclos de vida están íntimamente relacionados.
- No física o conceptual: El ciclo de vida de los elementos son independientes uno del otro.

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

Cuando un objeto no existe sin el otro.

117. ¿Qué es una clase?

Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común. También se lo denomina plantilla o molde a partir del cual se crean objetos.

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

No, no la proporciona, lo hace 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, la proporciona la interfaz. 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?

Se puede dividir en tres partes:

- Public: (publica) accesible a todos los clientes.
- Protected: (protegida) accesible solo a la propia clase, subclases y sus clases amigas.
- Private: (privada) accesible solo a la propia clase y sus clases amigas.

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

Existen tres tipos:

- Generalización/especialización: Denota un “es-un” (por ejemplo: una rosa es un tipo de flor)
- Todo/parte: Denota una relación “parte-de” (Por ejemplo: un pétalo es un tipo de flor, es una parte de ella)
- Asociación: Denota alguna dependencia semántica entre clases de otro modo independiente (Ejemplo: un picaflor de las flores)

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

Las relaciones son:

- Asociación.
- Herencia.
- Agregación.
- Uso.
- Instanciación/ejemplares.
- Metaclase.

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

Una asociación solo denota una dependencia semántica y no establece la dirección de esta dependencia, a menos que se diga lo contrario, ni establece la forma exacta en que una clase se relaciona con otra. Mediante la creación de asociaciones, se llega a plasmar quienes son los participantes en una relación semántica, sus papeles o roles y su cardinalidad

124. ¿Qué significa la cardinalidad en una relación?

La cardinalidad denota una asociación. Es la multiplicidad con que se relaciona una clase con otra. Es importante modelar la forma como los objetos se asocian entre sí

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

Puede existir:

- 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. La subclase contiene los atributos y métodos de la clase primaria.

127. ¿Cuántos tipos de herencia existen?

Existen dos tipos de herencia:

- 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 (o superclase). 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?

Se denomina clase abstracta y se crea con la idea de que las subclases añadan cosas a su estructura y comportamiento, implementando métodos.

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?

Una superclase o Metaclase.

134. ¿Qué es el polimorfismo?

Es la propiedad que indica, la posibilidad de que una entidad tome muchas formas. El polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento.

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

Cuando una clase posee métodos que comparten el nombre y se diferencian por su firma se llama sobrecarga.

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

A diferencia de Pascal, que se orienta a objetos, no se puede crear una jerarquía de clases para los diversos tipos de datos de telemetría , en vez de eso hay que definir solo un registro variante monolítico que abarca las propiedades asociadas con todos los tipos de datos. 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?

Existen dos formas de contención:

- Contención por valor o física: Se subdivide en valor y referencia.
- Contención por referencia o no física: El ciclo de vida de los elementos son independientes uno del otro.

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

El contenedor contiene el objeto en sí. Cuando creamos un objeto contenedor, se crean también automáticamente los contenidos. 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 no puede ser cíclica.

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. A diferencia de la física por valor, puede ser cíclica.

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.

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, todo objeto es instancia de una clase.

144. ¿Qué es una metaclase?

La metaclase es una clase cuyas instancias son, ellas mismas, clases. Los lenguajes como Smalltalk y CLOS soportan el concepto de metaclase directamente. En cambio C++ no puede soportarlo. La metaclase lleva a la idea del modelo de objetos a su conclusión natural en los lenguajes de programación orientados a objetos puros.

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

Las métricas que hay que observar para determinar la calidad de una abstracción son:

- Acoplamiento.
- Cohesión.
- Suficiencia.
- Compleción o estado completo.
- Ser primitivo.

146. ¿Qué es el acoplamiento?

Es el grado de dependencia entre los módulos que componen un sistema orientado a objetos. Es conveniente que sea baja debido a que un cambio puede impactar mucho en los demás módulos.

147. ¿Qué es la cohesión?

La cohesión también proviene del diseño estructurado, ésta mide el grado de conectividad entre los elementos de un solo módulo. La forma de cohesión menos deseable es la cohesión por coincidencia, en la que se incluyen en la misma clase abstracciones sin ninguna relación,

148. ¿Qué es la suficiencia?

Significa que la clase o modulo captura suficientes características de la abstracción como para permitir una interacción significativa o eficiente.

149. ¿Qué es la compleción?

Significa que la interfaz de la clase o modulo captura todas las características significativa de la abstracción. Mientras la insuficiencia implica una interfaz mínima, la interfaz completa es aquella que cubre todos los aspectos de la abstracción.

150. ¿Qué significa ser primitivo?

Son aquellas que pueden implantarse eficientemente solo si tiene acceso a la representación subyacente de la abstracción.

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

Al momento de tomar una decisión de este tipo se considera:

- Reutilización o reusabilidad: ¿este comportamiento sería útil en más de un contextoft
- Complejidad: ¿Qué grado de complejidad plantea el implementar este comportamientoft
- Aplicabilidad: ¿Qué relevancia tiene este comportamiento para el tipo en que podría ubicarseft
- Conocimiento de la implementación: ¿depende la implementación del comportamiento de los detalles internos de un cierto tipoft

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

El paso de mensajes debe adoptar las siguientes formas:

- Síncrono.
- Abandono inmediato.
- De intervalo.
- Asíncrono.

153. ¿Qué características posee un mensaje síncrono?

Un mensaje síncrono es una operación que solo comienza cuando el emisor ha iniciado la acción y el receptor está preparado para aceptar el mensaje, el emisor y el receptor esperaran indefinidamente hasta que ambas partes estén preparadas para continuar.

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

Un mensaje de abandono inmediato, es igual que el mensaje síncrono, excepto en que el emisor abandonara la operación si el receptor no está preparado inmediatamente.

155. ¿Qué características posee un mensaje de intervalo?

Un mensaje de intervalo es igual que el síncrono, solo que el emisor esperara que el receptor esté listo solo durante un intervalo de tiempo especificado.

156. ¿Qué características posee un mensaje Asíncrono?

Un mensaje asíncrono es cuando el 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?

La ley de Demeter afirma que los métodos de una clase no deberían depender de ninguna manera de la estructura, salvo de la estructura inmediata de su propia clase. Cada método debería enviar mensajes a objetos pertenecientes a un conjunto muy limitado de clases

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

La consecuencia inmediata de aplicar ésta 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?

Existen cuatro formas fundamentales:

- 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 solicitud entre las cosas nos permite exponer lo que tienen en común en abstracciones clave y mecanismo y eventualmente nos lleva a arquitecturas más pequeñas y simples. No existe un camino trillado hacia la clasificación.

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

La clasificación es muy difícil ya que, cualquier clasificación, es relativa a la perspectiva del observador que la realiza y al ser, ésta misma, inteligente requiere una gran cantidad de imaginación creativa.

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

El rol es trascendental, ya que 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?

Las aproximaciones generales a la clasificación son tres:

- Categorización clásica.
- Agrupamiento conceptual.
- Teoría de prototipos.

165. ¿Qué es la categorización clásica?

La categorización clásica es cuando todas las entidades que tienen una determinada propiedad o colección son necesarias y suficientes para definir la categoría. Esta clasificación proviene en primer lugar de Platón y luego de Aristóteles, por medio de su clasificación de plantas y animales.

166. ¿Qué es el agrupamiento conceptual?

La clasificación por agrupamiento conceptual es la variación moderna del enfoque clásico y deriva en gran medida de los intentos de explica cómo se representa el conocimiento. Como afirman Stepp y Michalski *"en éste enfoque las clases se generan formulando primero descripciones conceptuales de estas clases y clasificando entonces las entidades de acuerdo con las descripciones"*

167. ¿Qué es la teoría de prototipos?

Esta clasificación se crea para 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?

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 por el cual son relevantes para el diseño y suprimen las cosas que están fuera del sistema, o sea son superfluas. Esta identificación es altamente específica de cada dominio.

169. ¿Qué son los mecanismos?

Los mecanismos se utilizan para describir cualquier estructura mediante la cual los objetos colaborasen para proporcionar algún comportamiento que satisfaga un requerimiento del problema. O sea, un mecanismo es una decisión de diseño cobre como cooperan colecciones de objetos. Los mecanismos representan así otros patrones de comportamiento.

UNIDAD II

170. ¿Qué es un Campo de una clase?

El campo de una clase es una propiedad o atributo.

171. ¿Qué es un método de una clase?

El método de una clase es un procedimiento.

172. ¿A qué denominamos sobrecarga?

Llamamos sobrecarga ccuando se tienen varios métodos con igual nombre y distinta firma. Firma es el conjunto nombre del método + cantidad de parámetros + tipo de parámetro. El tipo de valor devuelto en caso de que el método sea una función no interviene en la determinación de la firma.

173. ¿Qué es una propiedad de una clase?

La propiedad de una clase son los atributos que ésta posee.

174. ¿Qué tipos de propiedades existen?

Existen cuatro tipos de propiedades:

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

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

Se engloban en tres ámbitos:

- Public.
- Private.
- Friends.

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

Las características son:

- Public: Es visto por todas las clases
- Private: Es visto solo por la propia clase.
- Friends: Es visto por la misma clase y clases amigas.

177. ¿Para qué se utilizan los constructores?

Los constructores se utilizan para cuando se desea inicializar el estado de un objeto ni bien se crea. Cuando poseen parámetros, se les pueden pasar valores desde el exterior que sirvan para inicializar el estado o lograr un comportamiento peculiar en ese preciso momento inicial. Su forma se comienza con *Sub New* y *End Sub.*

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

El tiempo de vida de un objeto se extiende desde el momento en que se crea por primera vez (y consume espacio) hasta que el espacio se recupera. Para crearlo hay que declararlo o bien asignarle memoria dinámicamente. Declarar un objeto es crear una nueva instancia en el montículo (heap). El constructor es el que asigna espacio para el objeto y establece su estado inicial estable (*sub new y end new*)

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

No, no se utiliza un contador de referencia.

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

utilizan?

El encargado de liberar espacio cuando hay objetos que ya no se utilizan se llama Garbagge Collector.

181. ¿Qué son los sucesos?

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

182. ¿Qué se utiliza para declarar un suceso?

public delegate void CupoLLeno();
public event CupoLleno SeLlenoElCupo;

183. ¿Cómo se logra que ocurra un suceso?

Según el ejemplo de la pregunta anterior: SeLLenoElCupo();

184. ¿Cómo se pueden atrapar los sucesos?

Los sucesos se atrapan con: *WithEvents*

185. ¿Para qué se utiliza Addhandler en un suceso?

Utilizar 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)
- La dirección de la rutina que controla el suceso (en el formato AddressOfnombrerutina).

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

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

187. ¿Qué características poseen los campos compartidos?

Los campos compartidos son variables que pueden ser compartidos, o bien llamados, accesibles, por todas las instancias de una clase determinada. Hay que ponerle la palabra *Shared* adelante. Ejemplo: *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.

188. ¿Qué características poseen los métodos compartidos?

La característica que poseen los métodos compartidos es la palabra *Shared* y se utiliza para marcar un método como estático, lo que permitirá llamar al método sin necesidad de generar una instancia del objeto de dicha clase.

189. ¿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).

190. ¿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.

191. ¿Qué ámbitos / modificadores de acceso existen? Explique las características de cada uno

Existen cinco ámbitos de clases anidadas:

- **Público**: El ámbito público hace que una clase, o cualquiera de sus miembros, sea visible fuera del ensamble actual si el proyecto es un proyecto de "biblioteca"
- **Privado**: El ámbito privado convierte, justamente, 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.
- **Friend**: El ámbito 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.
- **Protected**: 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
- **Protected Friend**: Este ámbito 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.

192. ¿Qué cosas se pueden heredar?

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

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

El polimorfismo se utiliza para la reutilización del 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* (Redefinir)

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

La clase *Derived* se utiliza sin tener que conocer que la deriva de otra clase. Sin embargo, conocer la relación de herencia entre las dos clases ayuda a escribir un código más flexible.

195. ¿Qué representa la clase this?

Representa el contexto actual de la clase en ejecución.

196. ¿Qué clase representa a la clase base?

Representa la clase base de una jerarquía de clases. También llamado superclase.

197. ¿Para qué se usa una clase abstracta?

Una clase abstracta es aquella que puede heredarle a una o mas subclases, pero no puede ser instanciada. Esto trae aparejado que nunca podemos tener objetos de clases abstractas. Son creadas para contener definiciones que deseamos que futuras subclases posean.

198. ¿Para qué se usa una clase sellada?

Una clase sellada puede instanciarse, pero no se podrá heredar desde ella. Se denomina como: *Sealed.*

199. ¿Qué es la sobreescritura?

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

200. ¿Qué elementos se pueden sobreescribir?

Se puede sobreescribir los métodos.

201. ¿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 (tapar) 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 resuelve en favor del elemento que sombrea.

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

El sombreado se caracteriza porque sobreescribe 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 III

203. ¿Qué es un Framework?

Un Framework, también llamado entorno de trabajo, 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. Sus características principales son:

- Normalizar u homogeneizar para usarlo de forma común y colocarlo en el entorno.
- Estandarizar para generar documento.
- Reutilizar experiencias.

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

Los frameworks 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.

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

Los hot-spots o bien llamados puntos calientes, 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.

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

Un framework se puede clasificar de dos maneras:

- Framework de caja blanca.
- Framework de caja negra.

207. ¿Qué es un Framework de Caja fílanca?

En un framework de caja blanca también llamado framework con arquitectura de configuración/conducidos, solamente es posible a través de la creación de nuevas clases. Estas clases y el código se pueden introducir en el marco por herencia composición. Uno debe programar el framework y entenderlo muy bien para producir un caso.

208. ¿Qué es un Framework de Caja Negra?

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). Por ejemplo, es posible utilizara un Wizard o configurador gráfico que dirija al usuario gradualmente paso por paso, a través del proceso de instanciación del framework. La caja negra del framework no requiere que el usuario sepa los detalles internos del framework. Por lo tanto, estos frameworks también se llaman frameworks dato-conducidos y son generalmente más fáciles de usar.

209. ¿Qué ventajas posee utilizar un Framework?

Las principales ventajas de la utilización de un framework son:

- El desarrollo rápido de aplicaciones. Los componentes incluidos en un framework constituyen una capa que libera al programador de la escritura de código de bajo nivel.
- La reutilización de componentes software al por mayor. Los frameworks son los paradigmas de la reutilización.

- El uso y la programación de componentes que siguen una política de diseño uniforme. Un framework orientado a objetos logra que los componentes sean clases que pertenezcan a una gran jerarquía de clases, lo que resulta en bibliotecas más fáciles de aprender a usar.

210. ¿Qué problemas resuelve .NET Framework?

El objetivo de .NET es eliminar varios de los problemas que se le presentan a los desarrolladores. Antes de .NET pocas aplicaciones se ejecutaban en más de una plataforma de hardware y de software por lo tanto debían rescribir el software para adaptarlo a los distintos entornos. Otro de los problemas era la comunicación y el intercambio de datos entre distintas aplicaciones.

La plataforma .NET resuelve estos problemas utilizando el Common Language Runtime que es independiente de hardware y software y por medio de la utilización de XML como lenguaje intercambio de datos universal entre aplicaciones. Ahora los desarrolladores pueden **escribir** aplicaciones en cualquier lenguaje .NET y estar seguros que pueden ser ejecutadas en todas las plataformas de hardware y software compatibles con .NET.

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

El CLR es el motor de ejecución (Runtime) del.NET Framework y 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.

212. ¿Qué es el MSIL?

MSIL o Microsoft Inmediate Language, como bien lo dice, es un lenguaje intermedio común a todos los sistemas operativos que soporten .net framework. 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.

213. ¿Qué es el CTS?

Define un conjunto común de "tipos" orientado a objetos, los cuales son tres:

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

- Todo tipo hereda directa o indirectamente del tipo OBJECT.
- Todo tipo de VALOR y de REFERENCIA.

214. ¿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.

215. ¿Qué es una excepción?

Una excepción son acciones o 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. Tratamiento de excepción con *try, catch.*

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

Se coloca el código para el tratamiento de la excepción producida en el bloque *try*.

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

```
class MyException : Exception
{
}
```

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

Sino esta tratado de igual manera lo atrapa el framework. La excepción se propaga hasta llegar al nivel más superior.

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

El de mayor jerarquía es: *Exception.*

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

Se encuentra en: *System.Exception*

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

Las clases genéricas más importantes son:

- System.System.Exception y son la mayoría de los objetos *Exception* definidos en .NET Framework que heredan de aquí
- System.AplicationException Son los objetos de excepción personalizados y específicos de cada aplicación que heredan de aquí.

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

Se utilizan:

Try.
Catch.
End Try.

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

Se coloca el código protegido como*: Try/End Try*

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

Se utiliza: *DivideByzeroException.*

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

Se utiliza: *System.DLLNotFoundException.*

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

Se colocaría: *When.*

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

Se colocaría dentro del bloque *Finally* porque siempre se ejecuta.

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

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. Produce una excepción dentro de un procedimiento.

229. ¿Escriba el código que permitiría provocar una excepción del tipo "ArgumentException"?

Código que permitiría provocar una excepción del tipo "*ArgumentException*"
Try
Throw New ArgumentException
End Try

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

Class MiExcepcionPersonalizaException Inherits System.ApplicationException.

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

Try

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

End Try

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

Se encuentra en el Heap o "monton administrado"

233. ¿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?

Los dos métodos más notorios son los procedimientos denominados:

- *Finalize()*

- *Dispose*()

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

Finalize hereda la clase del: *System.Object.*

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

Finalize()

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

GC.Collect()

237. ¿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()

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

Finalize()

239. ¿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.

240. ¿Cómo obtengo el método "Dispose"?

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

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

El código para limpieza/ liberación de recursos.

242. ¿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.

243. ¿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, a esto llamamos Resurrección de Objetos

244. ¿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.

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

Hay sólo tres valores distintos de generación. Tomando en cuenta que 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)

246. ¿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.

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

Se obtiene desde: *GC.GetGeneration()*

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

Se recupera a partir de: *GC.GetTotalMemory()*

249. ¿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?

Se utiliza: *GC.SupressFinalize()*

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

Se solicita por medio de: *GC.SupressFinalize()*

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

Se solicita al: *GC.WaitForPendingFinalizers()*

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

Se obtiene el número a partir de: *GC.GetGeneration()*

UNIDAD IV

253. ¿Qué son y para que se usan los tipos genéricosft

Usar tipos genéricos permite que un mismo fragmento de código adapte su funcionalidad a una multiplicidad de tipos.

254. Enumere las ventajas de utilizar genéricos.

Se puede hacer que una lista reciba el tipo con el que trabajará dinámicamente, lo que hace a la rutina más flexible y reutilizable.

255. ¿A qué elementos se le pueden aplicar genéricosft
A las variables y listas se les puede aplicar Genéricos,

256. ¿Qué es LinQ to Objectft
Es un lenguaje Integrado de Consulta que sirve para poder hacer consultas sobre Colecciones de Objetos, Bases de Datos y Archivos XML.

257. ¿Qué es una expresión Lambdaft
Son funciones anónimas que se usan frecuentemente para crear delegados en LINQ. Son simples, sin ninguna declaración, es decir, sin modificadores de acceso, la declaración de valor de retorno y nombre.
Es una especie de taquigrafía que permite abreviar el método en el mismo lugar que se usará. Nos ahorra tiempo, espacio y además, es fácil de revisar el código.

Están en el lado derecho del operador =fl. Una expresión lambda devuelve el resultado de la evaluación de la condición.

258. Ejemplifique cómo se puede utilizar una expresión lambda para realizar una consulta

int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var firstSmallNumbers = numbers.TakeWhile((n, index) =fl n fl= index);

UNIDAD V

259. ¿Para qué se utilizan las interfaces?

Una interfaz se utiliza 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.

260. ¿Cómo se implementa una interfaz?

public interface ITocable

```
{
        void tocar();
}
```

261. ¿Se pueden heredar las interfaces?

Si, por supuesto. Las interfaces se heredan entre sí. Una interfaz puede heredar de una interfaz, o más de una, lo que representa un aspecto muy interesante en cuanto a la flexibilidad que lo otorga el programador. También las interfaces admiten anidamientos.

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

Si, se puede implementar.

263. ¿Para qué se utiliza la interfaz IComparable?

Se utiliza una interfaz IComparable para ordenar únicamente por un solo criterio de ordenamiento. Los objetos sólo pueden compararse de una forma.

264. ¿Para qué se utiliza la interfaz IComparer?

Ya que la mayoría de los objetos del mundo real se pueden comparar y ordenar atendiendo al contenido de diferentes campos o combinaciones de campos, se puede 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.

265. ¿Para qué se utiliza la interfaz ICloneable?

La interfaz ICloneable se utiliza para admitir la clonación, se 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 sucesivamenteft Se puede implementar la interfaz ICloneable 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.

266. ¿Para qué se utiliza la interfaz IEnumereable?

La interfaz IEnumereable se utiliza para enumerar 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.

267. ¿Para qué se utiliza la interfaz IEnumerator?

Se utiliza la interfaz Ienumerator para crear un numerador personalizado, y posee los siguientes métodos:

- Función 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.

268. ¿Qué es un delegado?

Un delegado es algo muy similar a un puntero en una función de C en el sentido que permite llamar a un procedimiento utilizando un puntero que apunte al propio procedimiento.
Permite delegar la responsabilidad al cliente.

269. ¿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. Como también así apuntar a un procedimiento estático o a un procedimiento de instancia.

270. ¿Qué se puede delegar?

Se pueden delegar:

- Eventos.
- Funciones(Métodos).

271. ¿Cómo construiría un delegado?

Se construiría de la siguiente manera:

public delegate void CupoLleno();

272. ¿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.

273. ¿Para qué sirve la multidifusión de delegados?

Sirve la multidifusión de delegados para enviar una llamada a más de un procedimiento.

UNIDAD VI

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

Al tener una máquina cliente, requiere un servicio de una máquina servidor, y éste realiza la función para la que está programado. 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.

275. ¿Qué significa pasar información batch?

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.

276. ¿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. Se dice que la información está online o en línea, cuando se encuentra disponible a través de Internet.

277. ¿Qué es un protocolo?

Un protocolo es un 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

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

Una red de área local utiliza el protocolo: *TCP/IP*

279. ¿Qué protocolo usa internet?

Internet utiliza el protocolo: *IP*

280. ¿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.

281. ¿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.

282. ¿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 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.

283. ¿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.

284. ¿Qué característica posee un socket sincrónico?

Un socket sincrónico que envía permanece bloqueado esperando a que llegue una respuesta del receptor antes de realizar cualquier otro ejercicio

285. ¿Qué característica posee un socket asincrónico?

Un socket asincrónica que envía continúa con su ejecución inmediatamente después de enviar el mensaje al receptor.

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

Se puede utilizar el objeto WebBrowser disponible para aplicaciones Windows forms. Dicho objeto posee el método webBrowser.Navigate("http://www.google.com.ar");

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

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).

288. ¿Qué puertos posee una PC?

Los puertos que una PC posee:

- Paralelo - Serie - USB – FireWire.

289. ¿Cómo funciona el puerto paralelo?

El puerto paralelo es un esquema de transmisión de datos, justamente, en paralelo un dispositivo que envía datos a otro a una tasa de cierto número de bits a través de cierto número de cables a un tiempo. Sería fácil pensar que un sistema en paralelo es X 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.

290. ¿Cómo funciona el puerto serie?

El puerto serie es 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.

291. ¿Cómo funciona el puerto USfí?

Por norma general los cables de datos USB tienen cuatro conectores. Dos de ellos tienen funciones de alimentación eléctrica y el par restante se utiliza para la transmisión de datos. Los datos se pueden enviar de tres maneras, por interrupciones, en bloques, o de manera sincrónica:

- La transmisión por interrupciones es la que suele utilizarse para la mayoría de periféricos como teclados y ratones, que no requieren un monitorea y conexión permanente en el tiempo. Así, además de no saturar el ancho de banda se mantiene reducido el consumo energético.

- La transmisión en bloques contempla la movilización de grandes cantidades de información, por lo que es el método que se utiliza para la transmisión de datos en memorias y discos que se conectan por USB.
- La transmisión isócrona es un tipo de conexión que contempla el uso permanente del ancho de banda. Requiere altas tasas de transmisión de datos y es el tipo de conexión que vemos en monitores que funcionan por USB

292. ¿Qué es la domótica?

La domótica 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 de un hogar se regulen automáticamente en función de las condiciones exteriores, consiguiendo de paso un considerable ahorro energético. En otras palabras, la domótica regula la integración de tecnología en el diseño inteligente o automatizado de un inmueble con funciones de información, entretenimiento, gestión energética, seguridad y búsqueda de soluciones a la medida y aplicaciones según sus necesidades.
El término domótica viene de la unión de las palabras domus que significa: *casa* en latín y tica de automática, palabra en griego que significa: *funciona por sí sola.*

GUÍA DE TRABAJOS PRÁCTICOS
UNIDAD I

1. Desarrollar una jerarquía de clases relacionadas por herencia y agregación que represente la estructura necesaria para identificar claramente los elementos encontrados en un sistema de calificaciones de una universidad. En cada clase detallar métodos y propiedades.

2. Defina una situación donde se desee llegar desde un estado inicia a uno final, por ejemplo, una cuenta bancaria que posee un saldo inicial y se la somete a una operación bancaria. Desarrollar la clase cuenta con las propiedades y métodos que considere pertinentes y demuestre como cambia el estado del objeto en la medida que se le realizan depósitos, extracciones y transferencias.

3. Genere una estructura de clases donde se pueda observar la herencia simple y se justifique utilizar el polimorfismo.

4. Genere una estructura de clases donde se pueda observar la herencia múltiple y se justifique utilizar el polimorfismo.

5. Desarrollar conjuntos de objetos reales donde para su agrupación y clasificación sea necesario aplicar la categorización conceptual, la clásica y el agrupamiento prototípico. Justifique.

UNIDAD II

1. Desarrollar un programa que posea una estructura de clases donde se puedan observar dos métodos y el constructor sobrecargados.

2. Desarrollar un programa que posea una estructura de clases donde se pueda observar una propiedad de solo lectura, una propiedad de solo escritura, una propiedad de escrituralectura, una propiedad de predeterminada y una propiedad con argumentos.

3. Desarrollar un programa que posea una clase que aplique el Finalize y el Dispose, explique para qué utilizó cada uno y qué debió suceder para que funcione.

4. Desarrollar un programa que posea una clase que posea propiedades, métodos y sucesos. Los sucesos deben ser atrapados por el método tradicional (withevents) y por delegados (AddHandler).

5. Desarrollar un programa que posea una clase que contenga al menos dos campos compartidos, tres métodos compartidos, un constructor compartido y dos sucesos compartidos.

6. Desarrollar un programa donde se observe claramente el uso del MyBase. Enfatice las particularidades al usarlo en los constructores y finalizadores. Demuestre en el mismo programa el uso de me. Establezca las diferencias y en qué caso se justifica utilizar cada uno.

7. Desarrollar un programa que posea una clase que implemente dos propiedades. La primera de ellas deberá permitir calcular la edad de una persona y la segunda retornar un valor boolean que sea verdadero si la persona puede votar. Una persona puede votar cuando posee 18 o más años. La propiedad que determina calcula la edad es sobreescribible. Una segunda clase hereda de esta y sobre-escribe la propiedad colocando una implementación errónea (p.e. un cálculo erróneo de la edad). En la implementación de la propiedad booleana utilice primero MyBase y observe que ocurre, tome nota, luego haga lo mismo con MyClass, tome nota. Finalmente compare los resultados.

8. Desarrollar un programa que posea al menos una clase abstracta, un método virtual, una clase sellada y una clase anidada.

UNIDAD III

9. Desarrollar un programa que genera varias instancias (una cantidad importante), verifique la memoria utilizada, pase el GC y vuelva a verificar el espacio de memoria. ¿Qué se observaft

10. Desarrollar un programa que genere una instancia, pierda la referencia a la misma y aplicando la técnica de “resurrección de objetos” logre obtener la referencia a ese mismo objeto.

11. Desarrollar un programa que aplique el concepto de manejo de errores. La estructura propuesta debe tener al menos 5 Catch y el finally.

12. Desarrollar un programa que aplique el concepto de manejo de errores. Generar un error personalizado por medio de una clase que herede de Exception y disparar el error con Throw.

UNIDAD IV

13. Desarrollar un programa que aplique el concepto genéricos a nivel de clase y en almenos dos métodos.

14. Desarrollar un programa que utilice Linq para realizar consultas.

15. Desrrollar un programa que utilice expresiones lambda.

UNIDAD V

16. Desarrollar un programa que implemente la interfaz IComparable. Genere un arrary de objetos y ordénelos.

17. Desarrollar un programa que implemente la interfaz IComparer. La clase que lo implementa deberá tener al menos cinco criterios de ordenamiento. Genere un arrary de objetos y ordénelos por cada criterio.

18. Desarrollar un programa que implemente la interfaz ICloneable. Clone el objeto que posee la interfaz y demuestre que la clonación funcionó.

19. Desarrollar un programa que implemente las interfaces IEnumerable e IEnumerator. Lo adaptado recórralo con el for each y muestre los resultados.

UNIDAD VI

20. Desarrollar un programa que utilizando socket permita escribir en una aplicación que se esté ejecutando en una PC de la red y lo escrito se pueda visualizar en la aplicación que está corriendo en otra PC de la red.

21. Desarrollar un programa que utilizando socket que emule un servidor de chat. También desarrollar el software cliente que se ejecutará en las Pc's que se conecten al servidor. Cada mensaje enviado por un cliente lo podrán recibir todos los demás. En el caso que un cliente solicite comunicación privada el mensaje se verá solo por los que participan de esta comunicación. Un cliente puede solicitar armar un grupo y el resto unirse a él, para que esto suceda el propietario del grupo deberá aceptarlos y tiene la potestad de desconectarlos. El servidor es quien modera todo y por allí pasan todos los mensajes.

22. Desarrollar un programa que haciendo uso del puerto paralelo y un emulador que permita enviar señales al mismo y recolectar las entradas que se producen en él.

GUÍA DE ABORDAJE BIBLIOGRÁFICO

UNIDAD I

Cardacci Dario y Booch, Grady. **Orientación a Objetos. Teoría y Práctica.** -- Buenos Aires, Argentina. Pearson Argentina, 2013. Capítulos 1 al 4.

Deitel Harvey M. Y Paul J. Deitel. **Cómo programar en C#**. Segunda edición. Pearson. México 2007. Capítulos 9 al 11

UNIDAD II

Cardacci Dario y Booch, Grady. **Orientación a Objetos. Teoría y Práctica.** -- Buenos Aires, Argentina. Pearson Argentina, 2013. Capítulos 5.1 al 5.6

Deitel Harvey M. Y Paul J. Deitel. **Cómo programar en C#**. Segunda edición. Pearson. México 2007. Capítulos 9 al 11

UNIDAD III

https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/exceptionhandling-statements

Balena, Francesco. **Programación avanzada con Visual Basic 2005.** -- México, DF: McGraw-Hill Interamericana, c2008. Capítulo 1.

Deitel Harvey M. Y Paul J. Deitel. **Cómo programar en C#**. Segunda edición. Pearson. México 2007. Capítulos 12

UNIDAD IV

Deitel Harvey M. Y Paul J. Deitel. **Cómo programar en C#**. Segunda edición. Pearson. México 2007. Capítulos 25

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/generics/

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/linq-andgeneric-types https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/statements-expressionsoperators/lambda-expressions

UNIDAD V

https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/interfaces

https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/delegates

UNIDAD VI

MSDN. Microsoft Developer Network.
http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.aspx

Cardacci Dario y Booch, Grady. **Orientación a Objetos. Teoría y Práctica.** -- Buenos Aires, Argentina. Pearson Argentina, 2013. Capítulos 5.7.

Deitel Harvey M. Y Paul J. Deitel. **Cómo programar en C#**. Segunda edición. Pearson. México 2007. Capítulos 23

Montefinal, Fabián H.; Cardacci, Darío G. **Conceptos básicos sobre electricidad: electrónica y puertos de la PC**. 2a.ed.-- Buenos Aires: Universidad Abierta Interamericana, c2006. 78 páginas