PROGRAMACIÓN ORIENTADA A OBJETOS

GUÍA DE REVISIÓN CONCEPTUAL

Carrera: Ingeniería en Sistemas Informáticos

Fuente: Orientación a Objetos – Grady Booch y Darío Cardacci

2025

Unidad 1: Objetos y clases

Clase 1: Introducción a la programación Orientada a Objetos

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

Un software de gran magnitud es complejo por los siguientes motivos:

- Dificultad al momento de definir el dominio del problema.
- Dificultad al gestionar un proyecto de desarrollo de software.
- La no estandarización de los componentes de software.
- Complicaciones debido a los nuevos requerimientos del cliente o cambios en las tecnologías empleadas.

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

Los cinco atributos de un sistema complejo son:

1) La complejidad es jerárquica donde un sistema complejo se compone de subsistemas.
2) La abstracción del sistema queda en manos del observador y esta puede variar entre diferentes observadores.
3) Los componentes deben ser de alta cohesión (una clase bien definida y que tenga sentido en el contexto del sistema) y la relación entre diferentes componentes deben ser de bajo acoplamiento (baja dependencia entre componentes).
4) Un sistema complejo está compuesto de pocas clases, pero estas pueden variar en combinaciones y disposiciones.
5) Un sistema complejo funcional tuvo que haber pasado por un sistema simple, es decir, se tiene que emplear la estrategia bottom-up, yendo de las abstracciones sencillas hasta relacionarlas y formar el sistema complejo.

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

La primera es la llamada jerarquía estructural que se lee como “parte de” y consiste en que una abstracción puede tener más jerarquía que otra. Por ejemplo, una abstracción “*Coche*” puede tener otra abstracción jerárquica que sea “*Rueda*” y esta a su vez puede tener “*Llanta*”.

La segunda es la llamada jerarquía de tipos que se lee como “es-un” y es una forma de heredar una abstracción con otra. Por ejemplo, una abstracción “*Alumno*” es una “*Persona*” donde “*Alumno*” comparte las propiedades y métodos de “*Persona*”.

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

Para enfrentar a la complejidad se necesita crear varios “objetos” de la abstracción o clase y reconocer las “responsabilidades de hacer” y las “colaboraciones que se espera recibir” para hacer lo que el sistema pide.

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

La descomposición algorítmica que se basa en la estrategia top-bottom usando la secuencialidad y funciones. La otra descomposición es la orientada a objetos que se basa en la estrategia bottom-up usando las clases y objetos.

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

La diferencia entre ambas es en cómo se aborda el problema, la algorítmica va de lo más general hacia lo más específico, en cambio, la orientada a objetos va de lo más específico hacia lo más general.

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

La abstracción cumple el rol de observar un objeto de un sistema complejo e identificar las propiedades cualitativas y cuantitativas relevantes y las acciones que hace en el sistema.

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

La jerarquía cumple el rol de descomponer las abstracciones hechas en otras más específicas. Existen dos jerarquías; la estructural y la de tipos. Además, estas dos jerarquías conforman la estructura de clases utilizada en OO.

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

Considero que el diseño orientado a objetos es evolutivo porque antes se usaba el paradigma estructurado para descomponer un sistema. A medida que los sistemas fueron escalando, el software no se podía mantener. Ante este problema se optó por el diseño OO para crear un software más mantenible y escalable para los futuros nuevos requerimientos.

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

El DOO maneja cuatro modelos básicos. El modelo lógico que se centra en la estructura de clases y objetos, y el modelo físico que se centra en la arquitectura de módulos y procesos del código. También existe el modelo estático que define la estructura de clases usando el diagrama de clases y el modelo dinámico que define el comportamiento que tiene el objeto durante la ejecución.

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.

12. ¿Qué es el diseño orientado a objetos?

El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña.

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 las 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 elementos secundarios en el modelo de objetos son:

- Tipos.
- Concurrencia.
- Persistencia.

16. Explique el significado de la abstracción.

La abstracción significa observar el dominio del problema e identificar clases que interactúen con el sistema complejo, las acciones que realiza y que propiedades cualitativas o cuantitativas que pueda tener. De esta forma se puede acortar este dominio empezando desde lo más básico hasta lo más complejo del sistema.

17. Explique el significado del encapsulamiento.

El encapsulamiento es el proceso de guardar propiedades de una abstracción que conforma su estructura y comportamiento. Además, sirve para separar la interfaz (vista exterior) contractual y su implementación (vista interior).

18. Explique el significado de la modularidad.

La modularidad es la propiedad que tiene un sistema que fue descompuesto en un conjunto de módulos cohesivos (grupo de abstracciones que guarden una relación lógica) y débilmente acoplados (minimizando la dependencia entre módulos).

19. Explique el significado de la jerarquía.

La jerarquía es una clasificación de abstracciones. Las jerarquías más importantes son la estructura de clases (parte de) y la estructura de objetos (es-un).

20. Explique el significado de la tipificación.

La tipificación permite que un objeto cumpla con las propiedades y métodos que deba tener definidos en la clase. Además, no se debe confundir tipo con clase porque una clase puede definir a varios tipos mediante la herencia.

21. Explique el significado de la concurrencia.

La concurrencia es la propiedad que distingue un objeto activo de uno que no está activo. Esto quiere decir que los objetos pueden interactuar entre sí de forma simultánea, como si cada objeto fuera un hilo de procesador.

22. Explique el significado de la persistencia.

La persistencia es la propiedad de un objeto de perdurar en la memoria principal, por más que su creador haya dejado de existir. Por lo tanto, resulta importante que cuando un objeto cumpla su ciclo de vida, se lo destruya para liberar el espacio que ocupa en la memoria principal.

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?

Para abstraer correctamente siendo observador se necesita ver el comportamiento esencial de un objeto. Existe un principio denominado mínima sorpresa donde se busca capturar el comportamiento completo de un objeto evitando sorpresas o efectos laterales que puedan arruinar la abstracción que se está realizando.

24. ¿A qué denominamos un objeto cliente?

El objeto cliente es aquel que usa los recursos del objeto servidor usando el contrato del servidor.

25. ¿A qué denominamos un objeto servidor?

El objeto servidor es aquel que brinda servicios a otros objetos cliente definiendo los contratos que serán usados.

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

Meyer denomina al modelo contractual a la responsabilidad que tiene el objeto servidor de garantizar lo que tiene que hacer con el objeto cliente, respetando la firma o parámetros del contrato.

27. ¿Qué establece un contrato entre objetos?

Un contrato entre objetos establece las suposiciones que puede hacer un objeto cliente acerca del comportamiento de un objeto servidor.

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?

Se denomina protocolo y esta es la vista externa del objeto donde se pueden usar cada uno de los contratos definidos que se encargan de alguna responsabilidad que pueda requerir el objeto cliente.

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?

El conjunto completo de operaciones que puede realizar un cliente sobre un objeto se denomina protocolo.

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

La invariancia es un concepto central en la abstracción porque permite que algún método de un objeto cumpla con el contrato. La invariancia es una condición booleana que permite saber si el contrato se está cumpliendo antes o después de la ejecución del método.

Clase 2: Clases y Objetos

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

Para cualquier operación asociada a un objeto se debe definir las precondiciones y las postcondiciones. Adicionalmente, el concepto de invariante sirve como condición booleana que nos alerta si se logró o falló en la ejecución del método, quiere decir que la invariante nos puede alertar durante la precondición o postcondición.

2. ¿Qué es una precondición?

Una precondición quiere decir cuando un objeto cliente puede cumplir o no con las condiciones para usar algún contrato de un objeto servidor.

3. ¿Qué es una postcondición?

Una postcondición quiere decir cuando un objeto servidor puede fallar o no en su operación, pudiendo no satisfacer con la necesidad del objeto cliente.

4. ¿A qué se denomina excepción?

La excepción ocurre cuando no se pudo satisfacer a la invariante en la precondición o postcondición del contrato. Para esto se puede utilizar algún manejador de excepciones como la sentencia Try Catch.

5. ¿A qué se denomina mensaje?

Se denomina mensaje a la operación que se realiza sobre un objeto y a su consecuente reacción. Es una terminología empleada por el lenguaje Smalltalk.

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

Sí, porque la abstracción se centra en el comportamiento observable de un objeto y el encapsulamiento se centra en el ocultamiento de la información que no contribuyen a sus tareas esenciales.

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

El elemento de un objeto que captura su vista externa se denomina interfaz.

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

El elemento de un objeto que captura su vista interna se denomina implementación o implantación.

9. ¿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.

Los tres conceptos se asocian porque el ocultar la estructura y métodos de una abstracción permite que tenga más seguridad con respecto a los demás objetos.

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

Los dos aspectos son para construir módulos cohesivos, agrupando abstracciones que guarden una relación lógica, y módulos débilmente acoplados, es decir, que cada módulo sea independiente del otro con el fin de agilizar las compilaciones en caso de complicaciones o cambios que se deban realizar al sistema.

11. ¿Para qué se utiliza la jerarquía?

La jerarquía se usa para reducir el dominio del problema del sistema complejo, esto se logra identificando varias abstracciones y las jerarquía que puedan tener. Estas jerarquías pueden ser por estructura de clases u objetos.

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

Las propiedades de comportamiento que puedan compartir una serie de entidades se la denomina herencia múltiple.

13. ¿Las clases implementan tipos?

Sí, las clases permiten darle forma a la abstracción en un tipo.

14. ¿Los tipos implementan clases?

No, los tipos son creados a partir de una clase. El tipo es una forma de abstracción del dato que se crea a partir de una clase.

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

A estos lenguajes se los denomina fuertemente tipados, por ejemplo, C#, Java o C++.

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

A estos lenguajes se los denomina débilmente tipados, por ejemplo, JavaScript o Python.

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

La ligadura temprana se refiere el momento en el que se asignan las variables y expresiones con un tipo, en este caso se hace en el momento de compilación.

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

La ligadura tardía se refiere al momento en el que se asignan las variables y expresiones con un tipo, en este caso se hace en el momento de ejecución.

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

No, la comprobación estricta de tipos permite que las variables y expresiones estén bien tipadas, la ligadura dinámica asigna una variable con el tipo en el momento de ejecución.

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

La característica que permite a diferentes objetos actual al mismo tiempo se denomina concurrencia.

21. ¿A qué se denomina concurrencia pesada?

En el contexto de procesos, una concurrencia pesada se refiere a un proceso manejado por el sistema operativo que dispone de su propio espacio de direcciones de memoria.

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

En el contexto de procesos, una concurrencia liviana se refiere a un conjunto de procesos livianos manejados por el sistema operativo donde todos ellos comparten un espacio de direcciones de memoria.

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

La concurrencia es la propiedad que distingue de un objeto activo de uno que no lo está. Cuando un objeto está activo, se refiere a que ese objeto está representando un hilo de ejecución, es por esa razón que varios objetos se pueden comunicar de forma concurrente.

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

La característica en OO que permite conservar el estado de un objeto en el tiempo y el espacio se denomina persistencia.

25. ¿Qué cosas se persisten?

Se persiste el estado de un objeto, es decir, sus propiedades y métodos. Esta persistencia puede ser de tiempo, cuando el objeto sigue existiendo cuando su creador deja de existir, o puede ser de espacio, cuando el objeto cambia en las posiciones de memoria.

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

Desde la perspectiva de la cognición humana, un objeto es una cosa tangible o visible, algo que puede comprender intelectualmente y algo hacia lo que se dirige un pensamiento o acción.

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

Un objeto puede ser real o abstracto, y debe estar claramente definido para el dominio del problema del sistema.

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

Los objetos poseen límites físicos precisos, indicando que un objeto es algo que puede ser visto como un elemento independiente para ser analizado. Determinar el límite de un objeto puede ser una tarea complicada porque depende del observador que puede interpretar diferentes límites a un objeto.

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

Las tres cosas que debe tener un objeto son; estado, comportamiento e identidad.

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

La palabra instancia se puede usar como sinónimo de objeto, una instancia es un objeto creado a partir de una clase.

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

La palabra objeto se puede usar como sinónimo de instancia, un objeto es la instancia de una clase.

32. ¿Cómo definiría el estado de un objeto?

El estado de un objeto son todas las propiedades (normalmente estáticas) del mismo más los valores que puede tener (normalmente dinámicas) de cada una de las propiedades en un determinado tiempo.

33. ¿A qué definimos propiedad de un objeto?

Una propiedad es una característica distintiva, un rasgo o cualidad que ayuda a diferenciar un objeto del otro. Dichas propiedades de un objeto pueden tomar un valor, por ejemplo, la propiedad nombre, apellido y DNI de la clase Alumno puede tener los valores de José Pérez 44333111.

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

La distinción entre ambos es que los valores simples no varían con el tiempo y son inmutables, en cambio los objetos pueden variar con el tiempo, necesitan ser instanciadas y pueden crearse, destruirse y compartirse.

35. ¿Cómo definiría el comportamiento de un objeto?

El comportamiento es como actúa y cómo reacciona un objeto en función de sus cambios de estado y paso de mensajes con otros objetos.

36. ¿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?

Es correcto, el comportamiento de un estado se ve afectado por el estado del mismo. Por ejemplo, una máquina expendedora que puede soltar una bebida en función de que valores tenga su propiedad depósito.

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

Sí, los comportamientos entre objetos pueden modificar el estado de un objeto. Por ejemplo, en la máquina expendedora, se puede soltar una bebida en función del depósito que se le ingrese. Al realizar el método de soltar la bebida, el estado de la máquina expendedora se verá modificada, en específico, su propiedad de depósito.

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

Sí, es correcto afirmar que un objeto termina siendo los resultados acumulativos de su comportamientos, mediante el uso de métodos de uno de los objetos.

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

Una operación define un servicio que una clase ofrece a sus objetos clientes.

40. ¿Cuáles son las tres operaciones más comunes?

Las tres operaciones más comunes son el modificador, selector e iterador.

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

Las dos operaciones que se usan para crear o destruir una instancia son el constructor y el destructor.

42. ¿Qué tipo de operación es el modificador?

El modificador es un tipo de operación común que se encarga de alterar el estado de un objeto. Por ejemplo, para la clase Cola, pueden existir los modificadores de Agregar, Borrar y Modificar.

43. ¿Qué tipo de operación es el selector?

El selector es un tipo de operación común que se encarga de acceder al estado de un objeto sin alterarlo. Por ejemplo, para la clase Cola, pueden existir los selectores de Longitud, EstaVacia o Posición.

44. ¿Qué tipo de operación es el iterador?

El iterador es un tipo de operación común que se encarga de acceder a todas las partes de un objeto con algún criterio de ordenamiento. Por ejemplo, una operación que se encargue de iterar todos los nodos de la clase Cola.

45. ¿Qué tipo de operación es el constructor?

El constructor es una operación habitual que crea un objeto e inicializa su estado.

46. ¿Qué tipo de operación es el destructor?

El destructor es una operación habitual que libera el estado de un objeto y destruye el propio objeto.

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

Se los suele denominar subprogramas libres, en C++, particularmente, se los llama funciones no miembro.

48. ¿Todos los métodos son operaciones?

Todos los métodos son operaciones, dicho método está implementado en una clase en particular.

49. ¿Todas las operaciones son métodos?

No todas las operaciones son métodos, las operaciones pueden estar declaradas por fuera de una clase.

50. 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?

Es conveniente dividir el protocolo de un objeto en grupos lógicos de comportamiento para definir los papeles que puede cumplir el objeto.

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

Los grupos lógicos pequeños de comportamiento de un objeto son denominados los papeles que cumple dicho objeto.

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

Las responsabilidades que posee un objeto son; el conocimiento que posee el objeto (sobre sus propiedades) y las acciones que puede llevar a cabo (sobre sus métodos).

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

Sí, es relevante el orden de las operaciones de un objeto porque dichas operaciones modifican el estado del objeto.

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

Se dice que un objeto se puede considerar como una máquina por el funcionamiento que tiene, es decir, un orden en las operaciones que afectan al estado del objeto en función del tiempo y los eventos de otros objetos.

55. ¿Qué es la identidad de un objeto?

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

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

Significan que ambas variables tienen el mismo valor, pero tienen referencias distintas.

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

Significa que ahora X va a estar apuntando al mismo objeto que Y. Cualquier cambio que se haga desde ambas variables, modificará al mismo valor.

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

Significa que, Y tiene una copia del estado de X, pero no de la referencia de X.

59. ¿Qué significa realizar una clonación superficial?

La clonación superficial es la copia entre dos objetos donde se copia el objeto principal, pero las referencias internas se comparten. Si uno de los objetos cambia el contenido de un dato referenciado internamente, el otro también lo verá afectado, ya que apuntan al mismo lugar en memoria.

60. ¿Qué significa realizar una clonación profunda?

Una clonación profunda es la copia completa de un objeto, incluyendo todo lo que contiene internamente. Se copian de forma independiente todas las referencias, por lo tanto, los objetos no comparten estado. Modificar un objeto, no afecta al otro.

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

El ciclo de vida de un objeto es el tiempo de vida de un objeto que abarca desde el momento en el que se crea por primera vez, tomando espacio en la memoria, hasta que ese espacio en la memoria se libera.

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

Para liberar el espacio ocupado por un objeto, se hace uso de la recolección de basura que se encarga automáticamente de liberar el espacio. En otros lenguajes, para liberar el espacio es necesario llamar a un destructor.

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

Los tipos de relaciones que existen entre los objetos son el enlace y la agregación. Adicionalmente, dichas relaciones son importante para el análisis OO y diseño OO.

64. ¿Cómo podemos definir al enlace entre objetos?

En enlace se define como una conexión conceptual entre objetos. Un objeto colabora con otros objetos a través de sus enlaces con estos.

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

Los mensajes entre dos objetos en una relación de enlace pueden ser unidireccional o bidireccional.

66. ¿Qué es un mensaje unidireccional?

Un mensaje es unidireccional cuando un objeto cliente invoca operaciones sobre un objeto servidor, pero el objeto servidor no puede operar sobre el objeto cliente.

67. ¿Qué es un mensaje bidireccional?

Un mensaje es bidireccional cuando un objeto cliente invoca operaciones sobre un objeto servidor y viceversa.

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

El objeto cliente inicia el paso de un mensaje entre dos objetos en una relación de enlace.

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

En una relación de enlace, el objeto puede tomar el rol de actor, servidor o agente.

70. ¿Qué significa que un objeto actúe como “Actor”?

Un objeto actor significa que puede operar sobre otros objetos, pero ningún otro objeto puede actuar sobre él.

71. ¿Qué significa que un objeto actúe como “Servidor”?

Un objeto servidor significa que no puede operar sobre objetos, pero si puede ser operado por otros objetos.

72. ¿Qué significa que un objeto actúe como “Agente”?

Un objeto agente significa que puede actuar sobre objetos y también puede ser operado por otros objetos. Es una combinación entre actor y servidor.

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

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

Las cuatro formas de visibilidad que puede poseer un objeto servidor respecto del cliente son:

- Objeto servidor global para el objeto cliente.
- Objeto servidor es un parámetro de algún método del objeto cliente.
- Objeto servidor es parte del objeto cliente.
- Objeto servidor es un objeto declarado localmente en algún método del objeto cliente.

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

76. ¿Cuáles son las posibles formas de sincronización?

Las posibles formas de sincronización son; secuencial, vigilado y síncrono.

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

Significa que el funcionamiento de un objeto pasivo está garantizado por un objeto activo. Al ser secuencial, no hay un choque de operaciones entre objetos.

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

Significa que los objetos activos (o hilos de control) deben garantizar la exclusión mutua cuando operen sobre un objeto pasivo.

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

Significa que el objeto pasivo es el encargado de garantizar la exclusión mutua cuando múltiples objetos activos operan sobre el mismo objeto pasivo.

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

El enlace es una relación de igual a igual o también cliente/servidor.

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

La agregación es una relación jerárquica de todo/parte.

82. ¿Qué tipo de jerarquía denota la agregación?

La agregación denota una jerarquía donde se va desde el todo hasta sus partes. Por ejemplo, un objeto Auto (el todo) que está compuesto por los objetos Rueda, Llanta, Motor, entre otros (sus partes).

83. ¿Qué otros nombres reciben el "todo" en una relación de agregación?

El "todo" recibe el nombre del agregado o el contenedor.

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

Sí, el conjunto de todas las "partes" forman al estado del "todo".

85. ¿Qué tipos de agregación existen?

Los tipos de agregación que existen pueden ser de contención física o sin contención física.

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

La agregación con contención física se caracteriza porque sus partes están contenidas físicamente en el todo. Por ejemplo, los objetos Ala, Motor, TrenAterrizaje son partes que están físicamente dentro del objeto Avion, que conforma el todo.

87. ¿Qué es una clase?

Una clase es un conjunto de objetos que comparten una estructura y un comportamiento comunes. En la clase se debe definir los comportamientos comunes de los objetos y los objetos se instancian a partir de lo definido en la clase.

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

No, la interfaz de la clase proporciona su visión externa, exponiendo las operaciones de dicho objeto para que pueda ser usado por otros objetos.

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

No, la implementación de la clase proporciona su visión interna, es decir, que engloba los secretos de su comportamiento. Este comportamiento puede ser modificado debido a la ejecución de las operaciones expuestos en la interfaz de la clase.

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

La interfaz se puede dividir en; public, protected y private.

Public es una declaración accesible para todos los clientes. Protected es una declaración accesible solo a la propia clase y sus subclases. Private es una declaración accesible solo a la propia clase y sus clases amigas.

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

Los tipos básicos de relaciones que existen entre clases son:

- Asociación: representa una dependencia semántica entre dos clases.
- Herencia: representa la generalización/especialización o es/un entre clases.
- Agregación: representa el todo/parte en las clases.

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

Se desprenden las siguientes relaciones:

- Uso: establece los enlaces entre las instancias de las clases.
- Instanciación: al igual que la herencia, soporta una generalización, pero de forma diferente.
- Metaclase: es la clase de una clase, este concepto permite tratar a las clases como objetos.

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

No, la asociación denota una dependencia semántica y una dirección bidireccional. En una relación de clases bidireccional, ambas clases se conocen.

94. ¿Qué significa la cardinalidad en una relación?

La cardinalidad significa la multiplicidad que hay en la relación de asociación.

Un ejemplo de esta relación, usando las clases Venta y Producto, puede ser la siguiente;

- Cada *Producto* puede tener una *Venta*.
- Cada *Venta* puede tener uno o más *Productos*.

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

Pueden existir tres cardinalidades:

- Uno a uno.
- Uno a muchos.
- Muchos a muchos.

96. ¿Qué es la herencia?

La herencia es una relación entre clases en la que una clase comparte la estructura o el comportamiento definido de otra clase.

97. ¿Cuántos tipos de herencia existen?

Existe la herencia simple y la herencia múltiple.

98. ¿A qué se denomina herencia simple?

La herencia simple se refiere a que una clase hereda el comportamiento de una sola clase.

99. ¿A qué se denomina herencia múltiple?

La herencia múltiple se refiere a que una clase hereda el comportamiento de varias clases.

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

La clase que no se espera tener instancias de ella y solo se usará para heredar a otras clases se denomina clase abstracta.

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

La clase a la que se espera tener instancias de ella se denomina clase baja o clase concreta.

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

El método de una clase abstracta que no posee implementación y fue pensado para que sea implementado en las subclases que lo heredan se denomina método virtual puro.

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

La clase más generalizada en una estructura de clases se denomina clase base.

104. ¿Qué es el polimorfismo?

Es un concepto de teoría de tipos en el que un nombre puede denotar instancias de muchas clases diferentes relacionadas en una jerarquía de herencia. Cualquier objeto polimórfico es capaz de responder a alguna operación de diferentes formas.

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

La clase que posee métodos con el mismo nombre, pero varía su firma se denomina sobrecarga o polimorfismo paramétrico.

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

Hay que evitar las sentencias if y switch que son mal usados para definir el comportamiento de un objeto, lo correcto sería aplicar bien el polimorfismo.

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

La agregación es una relación donde existe un todo, que quiere decir que va a servir como contenedor de las partes. Es una relación de todo/parte, donde las partes están contenidas por el todo. Por ejemplo, la clase ControladorTemperatura es el todo y Calentador es una parte de esta.

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

En la agregación existen las formas de contención física por valor y por referencia.

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

La contención física por valor tiene la característica de que el tiempo de vida entre el todo y las partes es compartida. Quiere decir que, si se instancia el todo, también se instancia las partes o si se destruye el todo, también se destruyen las partes involucradas.

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

La contención física por referencia tiene la característica de que el tiempo de vida no es el mismo entre el todo y las partes. Quiere decir que cuando se destruye el todo, las partes pueden seguir estando en memoria actuando de forma independiente.

111. ¿Qué es una relación de uso?

Una relación de uso se da entre clases objetos/servidor, donde la clase cliente necesita los servicios de la clase servidor. Este tipo de relación es parecida a la asociación, pero más refinada y restringida.

112. ¿Qué es la instanciación?

La instanciación es el proceso de crear un objeto a partir de una clase. Adicionalmente, se habla de clases parametrizadas o genéricas que consiste en que la clase va a trabajar con el tipo de dato que se le pase al momento de instanciarla.

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

Sí, todo objeto es una instancia de una clase.

114. ¿Qué es una metaclase?

Una metaclase es una clase donde sus instancias son otras clases. El objetivo de la metaclase es definir variables o inicializarlas en las instancias de la metaclase.

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

Las métricas para determinar si una abstracción está bien diseñada son:

- Acoplamiento.
- Cohesión.
- Suficiencia.
- Compleción (estado completo/plenitud)
- Ser primitivo.

116. ¿Qué es el acoplamiento?

El acoplamiento es la dependencia que hay entre un módulo y otro. Un acoplamiento alto significa que los módulos dependen de otros, esto dificulta comprender los módulos, modificarlos o corregirlos. Un acoplamiento bajo significa que los módulos necesitan poca dependencia de otros módulos, esto ayuda a reducir la complejidad de un sistema y permite comprender, modificar y corregir los módulos.

En definitiva, para las abstracciones conviene realizar un acoplamiento bajo.

117. ¿Qué es la cohesión?

La cohesión mide la relación que existe entre los elementos de un solo módulo o clase. Por un lado, se tiene la cohesión por coincidencia donde en un módulo se tienen abstracciones donde no hay relación. Por el contrario, se tiene la cohesión funcional donde la relación entre los elementos y la clase es muy alta, por lo tanto, se tiene el comportamiento de una clase bien definida y delimitada.

En definitiva, para las abstracciones conviene realizar una cohesión alta.

118. ¿Qué es la suficiencia?

La suficiencia quiere decir si una clase tiene la cantidad suficiente de características para ser usada por los clientes. No se recomienda agregar muchas características innecesarias, la forma de evitar esto es viendo que operaciones necesita realizar el cliente.

119. ¿Qué es la compleción?

La compleción o cuando la clase está completa significa que la interfaz de una clase abarca todas las características importantes.

Esta métrica es subjetiva ya que las operaciones de alto nivel pueden ser equivalentes a varias operaciones de bajo nivel que puede usar el cliente.

120. ¿Qué significa ser primitivo?

En el contexto de las operaciones de una clase, una operación primitiva es aquella que debe tener acceso a las características de una clase para que sea implementada de forma eficiente y correcta. Dichas operaciones deben ser simples y fundamentales, por ejemplo, Agregar, Modificar, Borrar o Buscar, el uso de estas operaciones permite realizar operaciones complejas.

Esta métrica es subjetiva debido a que una operación compuesta por primitivas puede causar problemas de rendimiento.

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

Para decidir si una abstracción debe implementar un determinado comportamiento o no, se debe observar las siguientes características:

- Reutilización: si el comportamiento es útil en otros contextos.
- Complejidad: grado de dificultad del comportamiento.
- Aplicabilidad: relevancia del comportamiento en la abstracción elegida.
- Conocimiento de la implementación: es el grado en que una operación necesita saber cómo está hecha internamente una clase para poder funcionar.

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

El paso de mensajes entre objetos puede adoptar las siguientes formas:

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

123. ¿Qué características posee un mensaje síncrono?

Un mensaje síncrono significa cuando un emisor comienza la operación y el receptor está preparado para aceptar el mensaje. Ambos esperan indefinidamente hasta que estén listos y puedan continuar.

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

Un mensaje de abandono inmediato significa cuando un emisor comienza la operación y el receptor está preparado para aceptar el mensaje. El emisor cancela la operación si el receptor no está preparado de forma inmediata.

125. ¿Qué características posee un mensaje de intervalo?

Un mensaje de intervalo significa cuando un emisor comienza la operación y el receptor está preparado para aceptar el mensaje. El emisor espera la respuesta del receptor bajo un intervalo de tiempo, si se excede el tiempo, se cancela la operación.

126. ¿Qué características posee un mensaje asíncrono?

Un mensaje asíncrono significa que el emisor inicia la operación independientemente de si el receptor está esperando o no el mensaje.

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

Una abstracción está accesible a otra cuando es visible por otra abstracción y sus operaciones están en la vista externa para ser usadas.

128. ¿Qué expresa la Ley de Demeter?

La Ley de Demeter expresa que los métodos de una clase no deben depender de otras clases, excepto por las clases superiores o ella misma. Adicionalmente, los métodos deben ir dirigidos a objetos que pertenecen a un grupo limitado de clases.

129. ¿Cuál es la consecuencia inmediata de aplicar la Ley de Demeter?

La consecuencia de aplicar la Ley de Demeter es tener clases que son débilmente acopladas.

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

Las cuatro formas fundamentales donde un objeto X puede hacerse visible a un objeto Y son:

- 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.

131. ¿Para qué sirve clasificar a los objetos?

Clasificar objetos sirve para agrupar objetos que tienen estructuras o comportamientos comunes. Esto facilita el diseño, permite reconocer patrones en la interacción de objetos, organizar jerarquías (como generalización o especialización), mejorar la modularidad y guiar decisiones sobre cohesión y acoplamiento.

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

Es difícil la clasificación de objetos por dos razones:

- No existe una clasificación “perfecta”, la clasificación es relativa a la visión del observador que hizo las abstracciones.
- Requiere creatividad y originalidad para identificar relaciones y poder clasificar abstracciones.

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

El rol del observador es clave, ya que toda clasificación depende de su perspectiva. No hay una única forma correcta de clasificar objetos, porque diferentes observadores pueden ver y organizar el mismo conjunto de cosas de formas distintas según su enfoque o intereses.

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

Las aproximaciones generales a la clasificación son:

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

Estas categorizaciones son el fundamento para el análisis OO y permite usar estas prácticas y reglas que se pueden usar para categorizar clases y objetos en el diseño de un software complejo.

135. ¿Qué es la categorización clásica?

La categorización clásica es una forma de clasificar objetos en grupos que tengan propiedades comunes. Dicha categorización depende mucho del observador o diseñador y la forma de refinar dicha categorización es obteniendo más información del entorno.

136. ¿Qué es el agrupamiento conceptual?

El agrupamiento conceptual se trata de agrupar a las clases mediante una descripción conceptual, en vez de una propiedad. Esto provoca que una clase pueda ir a un grupo en base a que tanto se adecúa con el concepto y, además, puede estar en varios grupos a la vez.

137. ¿Qué es la teoría de prototipos?

La teoría de prototipos trata de agrupar clases teniendo en cuenta a un objeto prototipo. Si la clase se parece al objeto prototipo se lo clasifica de una manera, si no se parece, se lo clasifica de otra manera independientemente de si las propiedades o conceptos son parecidos.

138. ¿Qué es una abstracción clave?

Una abstracción clave es una clase u objeto que pertenece al vocabulario del dominio del problema y permite delimitar el problema lo máximo posible.

Para identificar esta abstracción es necesario el descubrimiento y la invención. Para el descubrimiento se necesita hablar con expertos del dominio del problema (por ejemplo, clientes, gerentes, empleados, etc.) e identificar qué clases suelen nombrar con frecuencia. La invención trata de hallar otras clases que complementen a la abstracción clave, estas no son fundamentales para el dominio del problema.

139. ¿Qué son los mecanismos?

Los mecanismos son decisiones de diseño que definen cómo colaboran los objetos entre sí para cumplir un comportamiento del sistema. Representan patrones de cooperación y permiten reutilizar estructuras de colaboración entre clases.

Existe una diferencia entre el diseño de una clase que tiene que ver con el comportamiento que tiene un objeto individual y el mecanismo, que plantea un diseño para saber cómo deben cooperar una colección de objetos.

Clase 3: Características de los objetos

1. ¿Qué es una propiedad de una clase?

Una propiedad es un miembro que proporciona un mecanismo flexible para leer, escribir o calcular el valor de un campo privado. La propiedad se puede utilizar como si fueran miembros de datos públicos, pero en realidad son métodos denominados descriptores de acceso. Existen dos descriptores de acceso; get (lectura) y set (escritura).

2. ¿Qué tipos de propiedades existen?

Existen los siguientes tipos de propiedades:

- Propiedad de lectura y escritura (get y set).
- Propiedad de solo lectura (get).
- Propiedad de solo escritura (set).

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

Los campos, métodos y propiedades de las clases pueden tener los siguientes ámbitos; public, private, protected, internal, protected internal o private protected.

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

Las características de cada ámbito según la visibilidad usada son:

- public: accesible desde cualquier lugar.
- private: accesible solo desde la misma clase.
- protected: accesible desde la clase y sus subclases.
- internal: accesible solo en el ensamblado (archivos .exe o .dll).
- protected internal: accesible desde el ensamblado o de una subclase de otro ensamblado.
- private protected: accesible desde el ensamblado y en la misma clase o en una subclase.

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

El tiempo de vida un objeto se refiere al momento en el que se construye (instanciación de la clase), se modifica su estado en un determinado tiempo y finalmente se destruye liberando su espacio en memoria.

6. ¿Qué es un campo de una clase?

El campo es una característica cualitativa o cuantitativa que posee la clase, dicha característica es interna a la clase. Para modificar un campo se necesita de una propiedad y un descriptor de acceso.

7. ¿Qué es un método de una clase?

El método de una clase es la operación que puede realizarse sobre el objeto. El método tiene un bloque de código, recibe un nombre, puede tener parámetros, puede retornar algún valor y puede ser de visibilidad public o private. El conjunto de todos estos elementos (excepto el valor de retorno) se denomina firma del método.

8. ¿A qué denominamos sobrecarga?

Se denomina sobrecarga a la creación de dos o más miembros en la misma clase con el mismo nombre que solo difieren en el número o tipo de parámetros. Se puede realizar la sobrecarga de métodos, constructores y propiedades indizadas.

9. ¿Qué tipos de parámetros puede tener un método?

Un método puede tener dos tipos de parámetros según si es Value Type o Reference Type.

En el caso de los Value Type, el funcionamiento de los tipos es el siguiente:

- Parámetros por valor: el método crea una copia del valor pasado como parámetro, si se modifica dicha copia dentro del ámbito del método, no se modifica el valor original.
- Parámetros por referencia: el método usa la referencia del valor pasado como parámetro (en C#, se lo indica usando “ref”), si se modifica el valor de la referencia en el ámbito del método, también se modifica el valor original.

En el caso de los Reference Type, el funcionamiento de los tipos es el siguiente:

- Parámetros por valor: el método crea una copia de la referencia del objeto. En el caso de querer asignar al parámetro a otra ubicación de memoria, no afectará al objeto original y creará otro objeto.
- Parámetros por referencia: el método crea una copia de la referencia del objeto. En el caso de querer asignar al parámetro a otra ubicación de memoria, dicha asignación afectará al objeto original.

10. ¿Para qué y cómo se usan los modificadores de parámetros in y out en los métodos?

El modificador de parámetro “in” se usa para pasar un parámetro al método en el que no se va a modificar su valor. Para usarlo es necesario poner en el argumento del método la palabra “in”, para la llamada es necesario que el parámetro pasado ya esté inicializado.

El modificador de parámetro “out” se usa para retornar más de un valor en un método. Para usarlo es necesario poner en el argumento del método la palabra “out”, para la llamada también se debe colocar la palabra “out”, pero no es necesario inicializar la variable porque dicha variable será asignada por el método.

Clase 4: Constructores y finalizadores

1. ¿Para qué se utilizan los constructores?

Los constructores son métodos que se usan para crear la instancia de una clase. Una clase puede tener varios constructores con diferente firma.

2. ¿Qué modificadores se le pueden colocar a los constructores y cómo los afecta en su funcionamiento?

A un constructor se le puede colocar el modificar público o privado. Si el constructor es público, entonces la clase permite crear una instancia. Si el constructor es privado, entonces la clase no permite crear una instancia.

3. ¿Para qué se utilizan los destructores?

Los destructores se utilizan para destruir instancias de una clase. Dicho destructor debe ser llamado por el programador y es una forma eficiente de manejar el espacio de memoria liberando los recursos administrados y no administrados.

Los recursos no administrados pueden ser la conexión a una base de datos, lectura de archivos con StreamWriter, etc.

Los recursos administrados puede ser un objeto al cual nadie está apuntando, por lo tanto el objeto está perdido en el espacio de memoria.

4. ¿Qué diferencia conceptual existe entre un finalizador y un destructor?

La diferencia conceptual entre finalizador y destructor es que en el finalizador se liberan los recursos no administrados y es llamado por el GC automáticamente para liberar los recursos administrados. En cambio, el destructor libera los recursos administrados y no administrados y es llamado mediante el método Dispose() por el programador en cualquier parte del código.

5. ¿Cómo desarrollaría un finalizador?

Un finalizador se coloca dentro de una clase y se lo pone con la siguiente sintaxis: ~Cliente() {MessageBox.Show(“El objeto ha sido destruido."); }

En dicho finalizador, en vez del MessageBox, se puede especificar que recursos no administrados se deben cerrar para esa clase.

6. ¿Cómo desarrollaría un destructor?

Para usar un destructor se debe utilizar la interfaz IDisposable y su método Dispose() para liberar el recurso de manera más controlada. Dentro del método Dispose() se especifica que recursos se deben liberar.

Adicionalmente, se pueden realizar llamadas al método Dispose(), a diferencia del otro finalizador donde no se puede llamarlo.

7. ¿Quién invoca al finalizador?

Nadie invoca al finalizador, el funcionamiento consiste en que el recolector de basura (o Garbage Collector) libera los objetos que no tienen referencia y después verifica el finalizador de la clase que podría liberar recursos no administrados por el GC.

8. ¿Quién invoca al destructor?

El programador es quién invoca al destructor llamando al método Dispose(), ofrece una forma más controlada de liberar recursos administrados y no administrados, en especial la lectura de archivos o conexiones a bases de datos.

9. ¿Una clase puede poseer un finalizador y un destructor?

Sí, una clase puede tener un finalizador y un destructor. El fin es para que otros programadores puedan utilizar Dispose() para liberar los recursos, en caso de que no lo utilicen, se ejecutará automáticamente el finalizador que libera los recursos no administrativos.

Para usar un finalizador y un destructor se debe de utilizar el patrón Dispose() donde se pasa un valor booleano, permitiendo ejecutar el Dispose() o el finalizador y no ambos al mismo tiempo. Es una forma prolija de controlar el ciclo de vida de los objetos y los recursos no administrados.

10. ¿Una clase puede tener muchos constructores?

Sí, una clase puede tener muchos constructores con diferente firma. Esto ofrece más flexibilidad para instanciar una clase, pudiendo elegir entre diferentes constructores.

Unidad 2: Objetos y clases – Relaciones

Clase 5: Eventos

1. ¿Qué son los sucesos?

Los sucesos o eventos es un mecanismo que permite que un objeto reaccione ante un estímulo externo. Es un mecanismo de enlace tardío que proporciona versatilidad a los desarrollos.

2. ¿Qué se utiliza para declarar un suceso?

Se utiliza la visibilidad que va a tener el evento, la palabra reservada “event”, el tipo EventHandler y por último el nombre del evento.

Esta sería su sintaxis:

public event EventHandler CambioEnNombre;

3. ¿Cómo se logra que ocurra un suceso?

Para que un evento ocurra se necesita realizar el desencadenamiento del evento. Para esto se debe indicar en qué parte de la clase se debe desencadenar el evento, esto puede ser cuando se modifica alguna propiedad (descriptor Set) o en algún método.

Esta sería su sintaxis:

CambioEnNombre?.Invoke(this, null);

4. ¿Cómo se pueden atrapar los sucesos?

Para atrapar al evento se necesita realizar la suscripción al evento. En específico, se necesita que un objeto se suscriba a una función, donde esta se va a ejecutar cuando se dispare el evento. Por lo tanto, se necesita tener declarado el método que se va a disparar.

La sintaxis para la suscripción es la siguiente:

A.CambioEnNombre += FuncionCambioEnNombre;

La sintaxis para cancelar una suscripción es la siguiente:

A.CambioEnNombre -= FuncionCambioEnNombre;

5. ¿Cómo le indica a un evento que desea cambiar el tipo del argumento que por defecto es EventArgs?

Al evento se lo debe declara de forma distinta. Para lograr esto se necesita programar una clase que represente al evento con argumento personalizado, dicha clase debe heredar de EventArgs y por buenas prácticas el nombre de la clase debe finalizar con EventArgs.

Ahora bien, para declarar el evento se necesita indicar un genérico que será la clase recién creada.

Sintaxis de la declaración del evento:

public event EventHandler <DatosCambioEnNombreEventArgs> CambioEnNombre;

Y ahora para el desencadenamiento se usa el siguiente código, indicando los argumentos personalizados definidos en la clase DatosCambioEnNombreEventArgs:

CambioEnNombre?.Invoke(this, new DatosCambioEnNombreEventArgs(this.Nombre));

En el ejemplo anterior, el segundo parámetro se pasa como null, en este caso, al utilizar un evento con argumento personalizado, se pasa como parámetro la propiedad Nombre del objeto.

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

Se pueden compartir clases, campos, propiedades, métodos, eventos y constructores mediante el uso del modificar static. Esto quiere decir que los miembros mencionados van a ser accedidos a nivel de clase y no a nivel de objetos, por esa razón los miembros son “compartidos”.

7. ¿Qué características poseen los campos compartidos?

Los campos compartidos tienen la característica de que deben ser llamados escribiendo el nombre de la clase y el nombre del campo. Dicho campo compartido o estático puede declararse en una clase estática o no estática.

8. ¿Qué características poseen los métodos compartidos?

Los métodos compartidos tienen la característica de que deben ser llamados escribiendo el nombre de la clase y el nombre del método, no se necesita crear una instancia para llamarlo.

9. ¿Qué características poseen los sucesos compartidos?

Los eventos compartidos tienen la característica de que funcionan a nivel de clase y no a nivel de objetos o instancias. Esto quiere decir que puede ser usado para manejar notificaciones comunes a todas las instancias ya que todos los objetos se suscriben al mismo evento compartido.

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

Las clases anidadas consiste cuando se declara una clase dentro de otra clase, por defecto la clase anidada tiene un modificador de acceso privado. Su uso consiste en que la clase anidada brinda servicios a la clase que la contiene. Un ejemplo de esto es una clase anidada que brinda el servicio de ordenamiento a la clase contenedora usando la interfaz IComparer.

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

Existen los siguientes modificadores de acceso:

- public: el acceso no está restringido.
- protected: el acceso está limitado a la clase y a sus subclases.
- internal: el acceso está limitado al ensamblado .dll o .exe.
- private: el acceso está limitado al tipo que lo contiene.
- protected internal: el acceso está limitado al ensamblado actual o las subclases de la clase base.
- private proctected: el acceso está limitado a la clase base o sus subclases dentro del ensamblado.

Clase 6: Tipos de clases, herencia y polimorfismo

1. ¿Qué cosas se pueden heredar?

Una clase derivada puede heredar las propiedades y métodos de la clase base. En dicha clase derivada se pueden agregar más propiedades y métodos que puedan especificar aún más el comportamiento especializado.

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

El polimorfismo se puede aprovechar en la práctica definiendo un método abstracto o virtual en una clase abstracta, y en las clases que hereden de esta clase base pueden sobrescribir el método virtual o deben sobrescribir el método abstracto. El objetivo del polimorfismo es reutilizar el código evitando usar sentencias de decisión como if else o switch.

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

La clase derivada se utiliza para que reciba propiedades y métodos de la clase base, si es una clase abstracta, entonces puede sobrescribir los métodos, lo que permite un mejor desarrollo y sentido en el código.

La forma de hacer herencia en C# es la siguiente:

class Auto : Vehiculo { // código }

4. ¿Qué representa this?

La palabra reservada “this” hace referencia a la instancia actual de la clase donde se está utilizando.

5. ¿Qué clase representa a la clase base?

La clase abstracta representa a la clase base, dicha clase será usada para que las subclases puedan heredar el comportamiento de la clase abstracta.

6. ¿Para qué se usa una clase abstracta?

La clase abstracta se utiliza para definir propiedades, métodos virtuales o abstractos que puedan ser heredados por una o varias subclases. En una clase abstracta no se puede realizar instancias.

7. ¿Para qué se usa una clase sellada?

Una clase sellada se utiliza para evitar la herencia en esa clase, el propósito de esto es por el análisis y diseño de sistemas, o para proteger la implementación de dicha clase.

8. ¿Qué es la sobreescritura?

La sobreescritura consiste en que una clase derivada pueda pisar el código heredado de la clase base. En el código se representa esto utilizando la palabra reservada “override”, por ejemplo:

public override decimal PrecioConDescuento() { // código }

9. ¿Qué elementos se pueden sobrescribir?

Los elementos que se pueden sobrescribir son los métodos o propiedades, dichos miembros pueden ser virtuales, abstractos u override (en el caso de que haya una herencia transitiva).

10. ¿A qué se denomina sombreado de métodos?

El sombreado de métodos se refiere cuando se quiere que el método sea accedido por el tipo de la clase derivada. A diferencia del override, cuando se accede al método de la clase base, mediante el polimorfismo, se ejecuta el método de una clase derivada.

En un sombreado de método se debe tener el tipo de la clase derivada y después se puede llamar al método sombreado.

Unidad 3: Framework y manejo de excepciones

Clase 9: Framework

1. ¿Qué es un framework?

Un framework es un marco de trabajo ofrece a quien lo utiliza, una seria de herramientas que le facilitan la realización de tareas. Dicho framework puede contener una librería de clases, documentación, ayudas, ejemplos, tutoriales. Esto quiere decir que un framework incluye la experiencia sobre algún dominio de problema específico.

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

Los frozen-spots o puntos congelados son aquellas partes del código provistas por el framework.

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

Los hot-spots o puntos calientes son las partes del código que el programador coloca e interactúa con el framework.

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

Un framework se puede clasificar según su extensibilidad, es decir, que un framework se puede utilizar como una caja blanca o como una caja negra.

5. ¿Qué es un Framework de Caja Blanca?

Un framework de caja blanca quiere decir que el programador necesita conocer la vista interna del framework para utilizarlo. La forma de utilizarlo es mediante la creación de una nueva clase que herede de la clase propuesta por el framework o mediante la composición.

6. ¿Qué es un Framework de Caja Negra?

Un framework de caja negra quiere decir que el programador no necesita conocer la vista interna del framework para utilizarlo. La forma de utilizarlo es instanciando una clase propuesta por el framework o creando un script de configuración donde el framework se encarga de la funcionalidad de ese script.

7. ¿Qué ventajas posee utilizar un Framework?

Una ventaja que posee es que el programador no necesita crear una estructura global para el dominio del problema desde cero, para eso puede utilizar un framework que ofrece una solución al dominio del problema de manera parcial donde el programador puede completarlo de una manera más ordenada y estándar.

Otra ventaja es en el ahorro de tiempo que supone al programador, ya que no necesita plantear soluciones desde cero, sino que sigue la definición y estandarización que ofrece el framework.

8. ¿Qué problemas resuelve .NET Framework?

Los problemas que resuelve .NET Framework son la ejecución y desarrollo de aplicaciones. Los elementos de .NET que permiten esto son el CLR, BCL, ADO.NET y las interfaces de usuario (WebForms, WindowsForms, Consola).

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

El CLR o Common Language Runtime es un entorno administrado que permite brindar los servicios comunes de forma automática.

Dichos servicios son el cargador de clases, MSIL a código nativo, controlador de código, GC, motor de seguridad, motor de depuración, verificador de tipos, controlador de excepciones, manejo de hilos, interacción con COM y el soporte a BCL (librerías de clases base).

10. ¿Qué es el MSIL?

El MSIL o Lenguaje Intermedio de Microsoft es un conjunto de instrucciones independientes de la CPU que se puede convertir de forma optimizada a código nativo mediante el uso de uno o más compiladores JIT proporcionados por CLR.

11. ¿Qué es el CTS?

El CTS (Common Type System) es un sistema común de tipos que define la forma en la que los tipos deben ser declarados, utilizados y administrados por el runtime.

Se divide en dos categorías que son los Value Type y Reference Type, ambos derivan de System.Object que es el tipo base.

12. ¿Qué es el CLS?

El CLS o Common Language Specification es un conjunto de reglas que los lenguajes de programación deben seguir para ejecutarse sobre la CLR y hacer uso de sus servicios.

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

Las instancias de los objetos administrados por el GC se encuentran en el heap administrado. El motor de optimización de GC determina el mejor momento para revisar el heap y liberar aquellos objetos que no tengan referencias.

14. ¿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 que deben implementar las clases para trabajar correctamente con la recolección de elementos y matar las instancias administradas y no administradas son el Finalize y el método Dispose.

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

Las clases heredan el método Finalize de la clase base System.Object, por lo tanto, todas las clases pueden usar un finalizador.

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

La firma que implementa el método Finalize debe tener el carácter “~” más el nombre de la clase donde se está codificando y no debe tener parámetros. Dentro del método finalizador se deben liberar los recursos no administrados, aquellos recursos administrados son liberados automáticamente por el GC.

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

El método que se utiliza para que el GC recolecte los elementos no utilizados es;

GC.Collect();

Dicho método fuerza a que el GC revise en el heap administrado y verifique si hay objetos para liberar.

18. ¿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?

El método que se utiliza para suspender el subproceso actual hasta que el subproceso que se está procesando en la cola de finalizadores vacíe dicha cola se llama:

GC.WaitForPendingFinalizers();

19. Cuando se ejecuta el método collect del GC, ¿qué método se ejecuta en los objetos afectados?

Cuando se ejecuta el método Collect del GC, se ejecuta el método finalizador de los objetos afectados. Dicho método finalizador tiene le forma de:

~NombreClase() { // Liberación de recursos no administrados }

En caso de que no se encuentre el método finalizador, entonces el GC solamente libera los recursos administrados, dejando en memoria aquellos recursos no administrados.

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

Si una clase no posee destructor, el método que se debería de exponer en una clase bien diseñada sería el Dispose().

21. ¿Cómo obtengo el método “Dispose”?

Para obtener el método Dispose(), se debe implementar en una clase la interfaz IDisposable, de esa forma la clase se ve obligada a implementar el método de la interfaz denominada Dispose().

22. ¿Qué se programa en el método “Dispose”?

En el método Dispose() se programa la liberación de los recursos no administrados. La ventaja de utilizar Dispose() es que el programador puede llamarlo cuando quiera, a diferencia del GC que revisa periódicamente el heap para liberar la memoria.

23. ¿Se pueden combinar el uso de “Dispose” y “Finalize”?

Sí, se puede combinar el uso de Dispose() y Finalize. Para hacerlo de forma correcta se debe utilizar una variable booleana que controle que no se ejecuten ambos, liberando así dos veces el mismo recurso.

La combinación de ambos sirve en un entorno de varios programadores, donde el programador puede utilizar Dispose(), respetando las buenas prácticas, o puede no utilizarlo y por ende se ejecutará automáticamente el finalizador para que libere los recursos no administrados.

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

Se denomina resurrección de objetos a la situación donde el GC ejecuta el finalizador de una clase y en dicho finalizador se "revive" al objeto sin referencia, colocándole nuevamente la referencia, y de esa forma el GC no libera dicho recurso administrado. Esto es considerado una mala práctica porque no respeta el ciclo de vida de un objeto.

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

Se denomina generación cuando el GC intenta liberar los recursos administrados y los objetos sobreviven, incrementando así su generación.

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

Los valores que puede adoptar la generación de un objeto son numéricos y oscilan entre 0 y 2.

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

Para obtener el número de veces que se ha producido el GC para la generación de objetos especificada se usa el siguiente método:

GC.CollectionCount(Int32);

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

Para obtener el número de generación actual de un objeto se usa el siguiente método:

GC.GetGeneration(Object);

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

Para recuperar el número de bytes que se considera que están asignados en la actualidad se usa el siguiente método:

GC.GetTotalMemory(Boolean);

El valor booleano que se pasa por parámetro indica si se debe forzar una recolección completa o no.

30. ¿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?

Para que un objeto no sea considerado por el GC desde el principio de la rutina actual hasta el momento en que se llamó al método se utiliza el siguiente método:

GC.KeepAlive(Object);

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

Para solicitar que el sistema no llame al finalizador del objeto especificado se usa el siguiente método:

GC.SuppressFinalize(Object);

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

Para solicitar que el sistema llame al finalizador del objeto especificado, para el que previamente se ha llamado a SuppressFinalize se debe llamar al siguiente método:

GC.ReRegisterForFinalize(Object);

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

Para obtener el número máximo de generaciones que el sistema admite en la actualidad, se hace uso de la siguiente propiedad estática:

GC.MaxGeneration;

Clase 10: Manejo de excepciones

1. ¿Qué es una excepción?

Una excepción es una situación inesperada o no deseada que se presenta durante la ejecución del programa.

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

En un bloque Catch se coloca la información que desencadenó la excepción, si en el Catch se define una variable de excepción, entonces se puede mostrar el mensaje de la excepción para obtener más información.

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

Para construir un objeto del tipo Exception personalizado se debe crear una clase que herede de Excepcion y el nombre de dicha clase debe finalizar con un Exception para seguir las buenas prácticas.

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

Si en un bloque de código donde se produce la excepción el error no está siendo tratado, entonces el programa deja de ejecutarse y presenta un mensaje de error. Esto es realizado por el framework .NET.

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

El objeto de mayor jerarquía para el manejo de excepciones proviene de la clase Exception.

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

La clase Exception se encuentra en el namespace llamado System.

7. ¿Cuáles son las dos clases genéricas más importantes definidas en el Frame-Work además de Exception?

Las dos clases genéricas más importantes definidas en el framework, además de Exception, son DivideByZeroException y FormatException.

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

La instrucción que se utiliza para controlar e interceptar las excepciones es la instrucción try-catch.

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

El código protegido contra excepciones se coloca en el bloque try, en caso de que ocurra una excepción se atiende dicha excepción en el bloque catch mostrando el mensaje.

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

Para interceptar un error de división por cero se usa la excepción DivideByZeroException.

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

Para interceptar una DLL que tiene problemas al ser cargada se usa la excepción DllNotFoundException.

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

Para especificar una condición adicional en el bloque catch para que sea seleccionada usaría la palabra reservada "when" y la condición a evaluar.

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

Colocaría el código de limpieza y liberación de recursos en el bloque finally, quedando la estructura de manejo de excepciones como try-catch-finally.

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

Para provocar un error y que el mismo se adapte al mecanismo de control de excepciones se usa la instrucción throw.

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

```
try
{
        textBox1.MaxLength = -5;
}
catch (ArgumentException ex)
{
        MessageBox.Show(ex.Message);
}
```

16. ¿Cómo construiría un objeto del tipo “Exception” personalizado?

Para construir un Exception personalizado, se debe crear una clase que herede de Exception, por ejemplo:

```
class ValorVacioException : Exception
{
        string valor;
        public ValorVacioException(string pValor) => valor = pValor;
        public override string Message => $”La propiedad {valor} no puede estar vacio!”;
}
```

17. ¿Cómo armaría un “Catch” personalizado para que se ejecute cuando se dé la excepción “ClienteNoExisteException”?

```
try
{
        // código donde puede ocurrir la excepción
        throw new ClienteNoExisteException();
}
catch (ClienteNoExisteException ex)
{
        MessageBox.Show(ex.Message, “ERROR”);
}
```

En dicho ejemplo, se puede sobrescribir el método Message de ClienteNoExisteException para mostrar un mensaje personalizado del error.

Unidad 4: Interfaces y delegados

Clase 11: Interfaces

1. ¿Para qué se utilizan las interfaces?

Las interfaces se utilizan para definir propiedades, métodos y eventos que luego deberán ser implementadas en aquellas clases que implementen dicha interfaz.

2. ¿Cómo se implementa una interfaz?

Para implementar una interfaz en una clase se debe poner ":" luego del nombre de la clase y la interfaz a implementar.

3. ¿Se pueden heredar las interfaces?

Sí, las interfaces se pueden heredar. Esto quiere decir que una interfaz puede heredar las propiedades, métodos y eventos de otra interfaz. La clase que implemente dicha interfaz está tipada por ambas interfaces y debe implementar el contrato que pida la interfaz.

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

Sí, se puede implementar un tipo de polimorfismo peculiar usando las interfaces. El uso de interfaces permite una herencia más "flexible" porque una clase A que implementa una interfaz debe implementar el contrato de dicha interfaz, lo mismo ocurre si una clase B, que no pertenece a la jerarquía de herencia de la clase A, implementa dicha interfaz.

La interfaz define la firma del método sin implementación, las clases que implementan la interfaz serán las que deban implementar dicho método.

5. ¿Para qué se utiliza la interfaz IComparable?

La interfaz IComparable se utiliza para interactuar con el método Sort de la clase Array, con el objetivo de que un Array de un determinado tipo pueda ser ordenado bajo un criterio. El contrato de IComparable es el método CompareTo, dicho método debe ser implementado por la clase que la utilice.

6. ¿Para qué se utiliza la interfaz IComparer?

La interfaz IComparer se utiliza para interactuar con el método Sort de la clase Array, con el objetivo de que un Array de un determinado tipo pueda ser ordenado bajo un criterio o varios criterios simultáneamente. El contrato de IComparer es el método Compare.

Una forma de usar IComparer es definiendo una o más clases dentro de una clase, con el objetivo de ordenar una o más propiedades. Por ejemplo, en una clase Persona, se pueden generar clases anidadas de ordenamiento como NombreASC, NombreDESC, ApellidoASC, ApellidoDESC, etc., donde cada una de esas clases anidadas debe implementar la interfaz IComparer.

7. ¿Para qué se utiliza la interfaz ICloneable?

La interfaz ICloneable se utiliza para realizar la clonación de objetos, creando una nueva instancia de una clase con el mismo estado de una instancia existente. El contrato de ICloneable es el método Clone. La clonación realizada puede ser superficial o profunda.

8. ¿Para qué se utiliza la interfaz IEnumerable?

La interfaz IEnumerable se utiliza para admitir una iteración simple sobre una colección. El contrato de IEnumerable es el método GetEnumerator que retorna un objeto de tipo IEnumerator.

9. ¿Para qué se utiliza la interfaz IEnumerator?

La interfaz IEnumerator se utiliza para definir cómo se recorre una colección, elemento por elemento. El contrato de IEnumerator es la propiedad Current y los métodos MoveNext y Reset.

Clase 12: Delegados

1. ¿Qué es un delegado?

Un delegado es un objeto que hace referencia a uno o varios métodos. Esto es útil para que el programador tenga un mayor poder cuando desea derivar dinámicamente ejecuciones de piezas de código.

2. ¿A qué elementos se les puede delegar?

Los elementos que se les puede delegar son los métodos. La firma de dicho método debe ser igual a la firma del delegado. Adicionalmente, se le puede delegar un método o una función lambda, el uso de funciones lambda permite más versatilidad en el código.

3. ¿Qué se puede delegar?

Se puede delegar la ejecución de uno o más métodos. Esto quiere decir que se puede pasar como parámetro a un delegado que representa la ejecución de uno o varios métodos.

4. ¿Cómo construiría un delegado?

La forma de construir un delegado es la siguiente:

```
public delegate int Delegado(string pTexto);
```

En este ejemplo el delegado retorna un int y recibe como parámetro un string, esto quiere decir que los métodos que pueden ser delegados deben cumplir con la firma del delegado declarado.

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

Primeramente, un procedimiento de devolución de llamadas o callback es un mecanismo donde un método se pasa como parámetro a otro método.

Entonces, la forma de implementarlo sería la siguiente:

```
public delegate void MostrarMensaje(string pMensaje);
public void Procesar(string pTexto, MostrarMensaje pCallback)
{
        pCallback($"Resultado: {pTexto}");
}
```

```
// Método que será pasado como parámetro en el método Procesar
public void MostrarEnPantalla(string pMensaje)
{
        MessageBox.Show(pMensaje);
}

private void button1_Click(object sender, EventArgs e)
{
        string texto = textBox1.Text;
        Procesar(texto, MostrarEnPantalla);
}
```

6. ¿Para qué sirve la multidifusión de delegados?

La multidifusión de delegados sirve para llamar a múltiples métodos con una sola llamada al delegado. Esto quiere decir que un delegado tiene la referencia de varios métodos y cuando se lo llama al delegado se ejecutan los métodos referenciados en orden.

Esto se usaría para ejecutar varios métodos automáticamente cuando ocurre un evento o se cumple una condición. También sirve para centralizar la lógica relacionada, por ejemplo, para notificar a múltiples partes del sistema sobre algún suceso. Por último, se puede utilizar para evitar código duplicado, en lugar de llamar a varios métodos uno por uno, se agrupan todos en un solo delegado.

Unidad 5: Genéricos, LINQ y expresiones Lambda

Clase 13: Genéricos y LINQ

1. ¿Qué son y para qué se usan los tipos genéricos?

Los tipos genéricos son parámetros de tipo que se usan para maximizar la reutilización del código mediante el diseño de clases y métodos que difieren su implementación hasta que la clase se instancia o el método es utilizado por código de cliente.

2. ¿A partir de que versión de C# se pueden utilizar?

Se pueden utilizar los tipos genéricos a partir de la versión 2.0.

3. Enumere las ventajas de utilizar genéricos.

Las ventajas de utilizar genéricos son:

a. Maximiza la reutilización del código.
b. Soluciona a la limitación de tener que tipar los elementos a un nivel de abstracción alto, como puede ser object.
c. Permite detectar errores de tipado en tiempo de compilación (antes de ejecutar el código).

4. ¿A qué elementos se le pueden aplicar genéricos?

Se le puede aplicar genéricos a los siguientes elementos; interfaces, clases, métodos, eventos y delegados.

5. ¿Cuáles son los usos más comunes de los genéricos?

Los usos más comunes de los genéricos es la maximización para reutilizar el código, tipado seguro en tiempo de compilación, restringir los tipos que deben entrar en una clase genérica o mejora de rendimiento al evitar el boxing y unboxing en una colección.

6. ¿Qué aspectos trascendentes se deben considerar al crear clases genéricas?

Al crear clases genéricas se deben considerar los siguientes aspectos:

- Qué tipos generalizar en parámetros de tipo.
- Las restricciones que se deben aplicar a los parámetros de tipo.

7. ¿Cuál es la diferencia entre heredar de una clase genérica abierta y una clase genérica cerrada?

Cuando se hereda de una clase genérica abierta, se espera que el tipo sea cualquiera que pueda aceptar la clase base genérica. En cambio, en una clase genérica cerrada, se espera recibir un tipo ya definido como int o string.

8. ¿Por qué es importante colocar restricciones en los parámetros de tipo?

Es importante colocar restricciones en los parámetros de tipo para controlar qué tipos pueden entrar como parámetro. Esto permite detectar los errores de parámetros de tipo en tiempo de compilación.

9. ¿Cuáles son los tipos de restricciones que se le pueden colocar a un parámetro de tipo?

Los tipos de restricciones que se le pueden colocar a un parámetro de tipo son:

- where T : struct
- where T : class
- where T : new()
- where T : unmanaged
- where T : <nombre de la clase>
- where T : <nombre de la interfaz>
- where T : U

10. Ejemplifique cómo crearía un método genérico con un parámetro de tipo.

```
// Suponiendo que la restricción del parámetro de tipo permite Empleado y sus derivados
public string Mostrar(T pEmpleado)
{
        return $"{pEmpleado.Nombre} {pEmpleado.Apellido}"
}
```

11. ¿Qué es LINQ?

LINQ o lenguaje integrado de consultas en un conjunto de herramientas que permite realizar consultas a distintas fuentes de datos y su sintaxis es parecida a la de SQL.

12. ¿Qué orígenes de datos se pueden consultar con LINQ?

En LINQ se pueden consultar a los orígenes de datos de objetos, XML o bases de datos.

13. ¿Cuáles son las partes básicas de una consulta LINQ?

Las partes básicas de una consulta LINQ son las siguientes:

- Obtener la fuente de datos.
- Crear la consulta.
- Ejecutar la consulta.

14. ¿Qué especifica la consulta en una estructura de LINQ?

La consulta especifica los objetos a recorrer en una fuente de datos y los objetos que se seleccionarán mediante una condición.

15. Mencione al menos tres cláusulas (las más importantes) que se usan en una expresión de consulta LINQ.

Las tres cláusulas más importantes que se usan para armar una consulta LINQ son **from**, **where** y **select**.

16. Explique que hace cada cláusula enumerada en la pregunta anterior.

La cláusula **from** especifica la fuente de datos, la cláusula **where** aplica el filtro de condición y la cláusula **select** especifica los elementos devueltos. La ejecución de la consulta LINQ devolverá un valor del tipo IEnumerable<T>.

17. Dado que una expresión de consulta genera un IEnumerable, enumere y explique los métodos de extensión que posee IEnumerable (p.e Count).

Algunos de los métodos de extensión que posee IEnumerable son:

- .Count() → cuenta la cantidad de elementos.
- .ToList() → convierte un IEnumerable en una lista List<T>.
- .First() → devuelve el primer elemento.
- .Where(…) → filtra elementos que cumplan una condición.
- .Select(…) → proyecta cada elemento.
- .OrderBy(…) → ordena por un campo ascendente.
- .OrderByDescending(…) → ordena por un campo descendente.
- .Any() → devuelve true si hay al menos un elemento.
- .All(…) → verifica si todos los elementos cumplen una condición.
- .Sum(), .Average(), .Max(), .Min() → operaciones matemáticas sobre campos numéricos.

18. ¿Qué utilizaría para ordenar en una expresión LINQ?

En una expresión LINQ utilizaría la cláusula order by e indico si es ascendente o descendente con ascending o descending. Por ejemplo:

consulta = **from** cliente **in** clientes **orderby** cliente.Nombre **ascending select** cliente

19. ¿Qué utilizaría para lograr una unión entre dos orígenes de datos en una expresión LINQ?

En una expresión LINQ utilizaría la cláusula join. Por ejemplo:

// Si quiero realizar una unión entre cliente y distribuidor
consulta = **from** c **in** clientes **join** d **in** distribuciones **on** c.Tipo **equals** d.Tipo **select new** { Cliente = c.Nombre, Distribuidor = d.Nombre };

20. ¿Cómo se pueden generar nuevos tipos utilizando LINQ?

Para generar nuevos tipos utilizando LINQ se debe utilizar la cláusula select new. Por ejemplo:

consulta = **from** c **in** clientes **select new** { c.Nombre, c.Apellido, c.FechaNacimiento };

Clase 14: Expresiones Lambda

1. ¿Qué es una expresión Lambda?

Una expresión Lambda es una función anónima que se puede usar para crear delegados.

2. Ejemplifique cómo se puede utilizar una expresión Lambda para realizar una consulta.

En este ejemplo se usa una expresión Lambda para realizar una consulta usando el método de extensión .Where():

var consulta = clientes.Where(c => c.Nombre.StartsWith(“B”));

3. ¿Qué debo realizar para crear una expresión Lambda?

Para crear una expresión Lambda se debe especificar los parámetros de entrada (si las hay) en el lado izquierdo del operador Lambda “=>”, y se debe colocar la expresión o el bloque de instrucciones en el lado derecho.

Por ejemplo: x => x*x

4. Indique cómo puede declarar un delegado utilizando Func y qué significa cada elemento utilizado.

Esta es la forma de declarar un Func:

Func<int, int, bool> funcion = **(x, y) => x == y**;

En este ejemplo se declara el delegado Func que espera recibir dos parámetros de tipo int y su retorno debe ser de tipo bool.

5. Crear un delegado utilizando Func que posea tres parámetros de entrada y uno de salida. Los parámetros de entrada son: el primero *int*, el segundo *double* y el tercero *bool*. El parámetro de retorno es de tipo *bool*.

Delegado Func:

Func<int, double, bool, bool> funcion;

Unidad 6: Comunicación entre dispositivos y manejo de dispositivos

Clase 15: Socket, comunicación entre aplicaciones y manejo de dispositivos

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

Un esquema cliente/servidor se caracteriza por tener dos roles: el cliente, quién solicita los servicios y el servidor, quién brinda los servicios. En la actualidad, un nodo en la red puede cumplir con ambos roles.

Otra característica es que el esquema debe funcionar en una red de computadoras interconectadas y, además, deben implementarse protocolos para asegurar esta comunicación. Los protocolos que se usan para garantizar la comunicación en la red son IP, para identificar de manera lógica un nodo en la red, y TCP o UDP, para intercambiar segmentos o datagramas en la red.

Por último, se tiene la forma en la cual se pueden enviar los datos, que puede ser mediante el pasaje en modo batch o el pasaje en modo online.

2. ¿Qué significa pasar información batch?

Cuando se pasa la información en modo batch significa que se pasan los datos mediante lotes, cuando el lote ya está lleno de datos, entonces se inicia la conexión, se envía el lote y finalmente se cierra la conexión.

3. ¿Qué significa pasar información online?

Cuando se pasa la información en modo online significa que primero se inicia la conexión entre cliente y servidor, y posteriormente se van enviando los datos que se pueden llegar a producir hasta que se cierre la conexión por parte del cliente o servidor.

4. ¿Qué es un protocolo?

Un protocolo es un conjunto de reglas que se deben cumplir para garantizar que las computadoras puedan comunicarse entre sí.

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

En una red de área local se usa el protocolo IPv4 o IPv6 para identificar de manera lógica y única a un dispositivo final en la red. Por ejemplo, un dispositivo final ubicado en RRHH puede tener la dirección IPv4 de 192.168.3.10.

6. ¿Qué protocolo usa Internet?

Internet utiliza los protocolos de IP, TCP o UDP, típicamente se dice que Internet utiliza los protocolos TCP/IP para la transmisión de datos.

En específico, Internet usa IP para enrutar los paquetes entre nodos. Para la transmisión de datos se usan TCP; que garantiza que un segmento de datos llegue a un nodo final y es orientado a la conexión; y UDP, que es de máximo esfuerzo y no es orientado a la conexión.

7. ¿Qué hace el protocolo IP?

El protocolo IP (Internet Protocol) se encarga de enviar los datos en forma de paquetes entre nodos a través de una red. También identifica a cada nodo mediante una dirección IP.

8. ¿Qué ventajas tiene distribuir procesos?

Las ventajas de distribuir procesos entre varios nodos es que permite aumentar el rendimiento, mejorar la escalabilidad y garantizar la continuidad del servicio ante fallos. Un ejemplo es el servicio EC2 de AWS.

9. ¿Qué ventajas tiene distribuir almacenamientos?

Las ventajas de distribuir almacenamientos entre varios nodos es que permite mejorar la disponibilidad, aumentar la escalabilidad, proteger datos mediante copias de seguridad y el acceso rápido desde lugares distintos. Un ejemplo de esto es el uso de RAID para distribuir o aplicar redundancia a los discos duros.

10. ¿Qué es un socket?

Un socket es la combinación entre una dirección IP y un número de puerto que se utiliza para la transmisión de datos en una red. La dirección IP identifica al host y el número de puerto identifica el servicio al cuál quiere acceder.

Por ejemplo: 192.168.0.10:22 → Puerto 22 identifica al servicio SSH

11. ¿Qué característica posee un socket sincrónico?

Un socket sincrónico se caracteriza por enviar o recibir datos de forma bloqueante. El programa que usa el socket se detiene hasta que la operación se complete.

12. ¿Qué característica posee un socket asincrónico?

Un socket asincrónico se caracteriza por enviar o recibir datos de forma no bloqueante. Permite que el programa siga ejecutándose mientras la operación se realiza en segundo plano. Cuando la operación termina, se notifica al programa mediante un evento o callback.

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

El objeto WebBrowser de Windows Forms se utiliza para construir un navegador dentro de una aplicación. Permite mostrar páginas web, navegar entre ellas y manipular su contenido.

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

Los puertos de la PC son interfaces de E/S que permiten la interacción entre el hardware del sistema informático y los dispositivos periféricos. Funcionan como puntos de acceso que facilitan la transferencia de datos, control de dispositivos y en algunos casos la alimentación eléctrica.

15. ¿Qué puertos posee una PC?

Una PC puede tener diversos puertos según la funcionalidad. El puerto que se usa para la comunicación a nivel de LAN es el puerto RJ45. Aquellos puertos que se utilizan de forma general o para periféricos de propósito simple son los paralelo, serie y USB.

16. ¿Cómo funciona el puerto paralelo?

El puerto paralelo consta de ocho líneas de datos que se usan para transmitir un byte a la vez y esto conlleva a que los bits deben enviarse en forma sincronizada. El estándar IEEE 1284 define cómo armar un hardware que va a utilizar el puerto paralelo. Dicho estándar establece los conectores tipo A, tipo B y tipo C. Otro conector es el DB25.

17. ¿Cómo funciona el puerto serie?

El puerto serie consta de una línea de datos que se utiliza para transmitir bit a bit. Se debe conocer cuando inicia y finaliza una secuencia de bits, para tratar este problema, se utilizan protocolos. Existen tres formas de comunicación en serie; simplex, dúplex y full dúplex. Algunos estándares son RS-232, FireWire, Serial ATA y USB.

18. ¿Cómo funciona el puerto USB?

El puerto USB funciona como una interfaz serial dúplex que permite la comunicación entre una computadora y múltiples dispositivos esclavos (mouse, teclado, micrófono, etc.) mediante un protocolo estándar. Utiliza un sistema de transmisión en paquetes, soporta varias velocidades de transferencia como Low, Full, High y SuperSpeed. Por último, proporciona alimentación eléctrica a los dispositivos conectados, permitiendo la conexión en cadena mediante hubs.

19. ¿Qué es la domótica?

La domótica es la integración de tecnologías y sistemas electrónicos para automatizar y controlar de forma inteligente las funciones del hogar, por ejemplo, la iluminación, climatización, seguridad y electrodomésticos.