

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

- **La complejidad del dominio del problema:** Los usuarios normalmente tienen dificultad al expresar con precisión sus necesidades de una forma en la cual los desarrolladores puedan comprender. Los usuarios y desarrolladores tienen perspectivas diferentes sobre la naturaleza del problema y realizan distintas suposiciones sobre la naturaleza de la solución. Además, otra complicación es que los requisitos cambian frecuentemente durante su desarrollo, por lo que el sistema evoluciona.
- **La dificultad de gestionar el proceso de desarrollo:** Se intenta que el sistema en desarrollo sea lo más comprensible posible, pero a veces el sistema es muy grande por lo que puede contener cientos de miles de líneas de código divididas en módulos. De esta manera, es complicado comprender un sistema de gran envergadura, por lo cual se requiere un gran equipo de desarrollo que también puede estar ubicado en diferentes partes del mundo, lo que también genera complejidad.
- **La flexibilidad que se puede alcanzar a través del Software:** El software ofrece la flexibilidad máxima por lo que un desarrollador puede expresar casi cualquier clase de abstracción. Esta flexibilidad empuja al desarrollador a construir por sí mismo prácticamente todos los bloques fundamentales sobre los que se apoyan estas abstracciones de más alto nivel.
- **Los problemas que plantea la categorización del comportamiento de sistemas discretos:** Se intenta diseñar los sistemas con una separación de intereses, de forma que el comportamiento de una parte del sistema tenga mínimo impacto en el comportamiento de otra parte del mismo. Sin embargo, sigue dándose el hecho de que las transiciones de fase entre estados discretos no pueden modelarse con funciones continuas. Todos los eventos externos a un sistema de software tienen la posibilidad de llevar a ese sistema a un nuevo estado, y más aún, la transición de estado a estado no siempre es determinista. En las peores circunstancias, un evento externo puede corromper el estado del sistema, porque sus diseñadores olvidaron tener en cuenta ciertas interacciones entre eventos. Los eventos externos pueden afectar a cualquier parte del estado interno.

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

- **La complejidad toma forma de jerarquía.** Sistemas complejos que se componen de subsistemas relacionados que a su vez tienen sus propios subsistemas. Por lo cual, se ven los sistemas por partes. Relaciones entre las partes.
- **Componentes primitivos.** Quedan a la clasificación del observador.
  - Alta frecuencia de componentes: Involucra la estructura interna de estos.
  - Baja Frecuencia: Involucra la interacción entre los componentes.
  - Esto posibilita el estudio de cada parte de forma aislada.
    - ✓ Sistemas jerárquicos descomponibles: Se pueden dividir en partes identificables
    - ✓ Sistemas jerárquicos casi descomponibles: Partes no completamente independientes.
- **Sistemas con economía de expresión.** Los sistemas complejos tienen patrones comunes, los cuales pueden conllevar a la reutilización de componentes pequeños.

- **Sistemas complejos tienden a evolucionar durante el tiempo.** A medida que los sistemas evolucionan, objetos que antes se consideraban complejos ahora se consideran primitivos.

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

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

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

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

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

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

Las dos formas de descomposición son:

- **Orientada a objetos.**
- **Algorítmica.**

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

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

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

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

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

**Resalta la estructura y comportamiento comunes en el interior de un sistema.** Así, por ejemplo, en vez de estudiar cada una de las células de una hoja de una planta específica, es suficiente estudiar 1 de esas células, porque se espera que todas las demás se comporten de modo similar.

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

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

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

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

**11. ¿Qué es la programación orientada a objetos?**

Es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia. MUY IMPORTANTE: La POO utiliza objetos no algoritmos, como sus bloques lógicos de construcción fundamentales (la

jerarquía es parte de). Cada objeto es una instancia de alguna clase; y las clases están relacionadas con otras clases

por medio de relaciones de herencia (jerarquía de clases). Si faltan cualquiera de estos 3 elementos un programa NO

es orientado a objetos.

**12. ¿Qué es el diseño orientado a objetos?**

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

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

**13. ¿Qué es el análisis orientado a objetos?**

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

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

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

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

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

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

1. **Tipos**
2. **Concurrencia**
3. **Persistencia**

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

#### 16. Explique el significado de la abstracción.

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

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

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

esencial de un objeto de su implantación.

La abstracción consiste en captar las características esenciales de un objeto, así como su comportamiento. Por ejemplo, ¿Qué características podemos abstraer de los automóviles? O lo que es lo mismo ¿Qué características semejantes tienen todos los automóviles? Todos tendrán una marca, un modelo, número de chasis, peso, llantas, puertas, ventanas,

etc. Y en cuanto a su comportamiento todos los automóviles podrán acelerar, frenar, retroceder, etc. En los lenguajes de programación orientada a objetos, el concepto de Clase es la representación y el mecanismo por el cual se gestionan las abstracciones. La abstracción es uno de los medios más importantes mediante el cual nos enfrentamos con la complejidad inherente al software. **La abstracción es la propiedad que permite representar las características esenciales de un objeto sin preocuparse de las restantes características (no esenciales).** La abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación.

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

### **17. Explique el significado del encapsulamiento.**

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

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

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

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

### **18. Explique el significado de la modularidad.**

**“La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados”.**

Básicamente se trata de hacer lo posible por construir módulos cohesivos (agrupando abstracciones que guarden cierta relación lógica) y débilmente acoplados (minimizando

las dependencias entre los módulos). Así, los principios de abstracción, encapsulamiento y modularidad son sinérgicos. Un objeto proporciona una frontera bien definida alrededor de una sola abstracción, y tanto el encapsulamiento como la modularidad proporcionan barreras que rodean a esta abstracción.

El concepto de modularidad, se refiere a una organización en la que distintos componentes de un sistema de programación se dividen en unidades funcionales separadas.

La modularidad tiene cohesión (relaciones fuertes entre elementos. Tiene mucha especificidad, la característica principal es que es específico. Cuando algo es poco cohesivo no es muy reutilizable. Lo contrario a cohesión es acoplamiento) y débilmente acoplado (un módulo que no depende de otro módulo es muy reutilizable.). Muy

importante: *Lograr fuertes relaciones internas y muy pocas dependencias o relaciones externas.*

## **19. Explique el significado de la jerarquía.**

**“La jerarquía es una clasificación u ordenación de abstracciones”**

Las dos jerarquías más importantes en un sistema complejo son su estructura de clases (la jerarquía de clases) y su estructura de objetos (la jerarquía de partes).

La herencia es la jerarquía <<de clases>> más importante, es un elemento esencial de los sistemas orientados a objetos. Básicamente **la herencia define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida en una o más clases** (lo que se denomina herencia simple o herencia múltiple, respectivamente). La herencia representa así una jerarquía de abstracciones, en la que una subclase hereda de una o más superclases. Típicamente, una subclase aumenta o redefine la estructura y el comportamiento de sus superclases. La herencia implica una jerarquía de generalización/especialización, en la que una subclase especializa el comportamiento o estructura, más general, de sus superclases. A medida que se desarrolla la jerarquía de herencias, la estructura y comportamiento comunes a diferentes clases tenderá a migrar hacia superclases comunes. Por esta razón se habla a menudo de la herencia como una jerarquía de generalización/especialización. Las superclases representan abstracciones generalizadas, y las subclases representan especializaciones en las que los campos y métodos de la superclases sufren añadidos, modificaciones o incluso ocultaciones.

**Las jerarquías <<parte de>> describen relaciones de agregación.** La agregación **permite el agrupamiento físico de estructuras relacionadas lógicamente**, y la herencia permite que estos grupos de aparición frecuente se reutilicen con facilidad en diferentes abstracciones. La agregación plantea el problema de la propiedad.

## **20. Explique el significado de la tipificación.**

**“Los tipos son la puesta en vigor de la clase de los objetos**, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas”

Permite la agrupación de objetos en tipos. Un tipo es una caracterización precisa de propiedades estructurales o de

comportamiento que comparten una serie de entidades. La idea de congruencia es central a la noción de tipos

(ejemplo de unidades de medidas en física). La comprobación estricta de tipos impide que se mezclen abstracciones.

Los tipos permiten expresar las abstracciones de manera que el lenguaje de programación en el que se implantan puede utilizarse para apoyar las decisiones de diseño.

Tipos != de Clase (son distintos, no es lo mismo!!!)

Objetos de diferentes tipos no se puedan intercambiar, o se pueden intercambiar solo de forma restringida. Tipo es una caracterización precisa de las propiedades estructurales y de comportamiento que comparten una colección de entidades. Existen lenguajes fuertemente tipificados (Ada) y débilmente tipificados. Estos últimos soportan polimorfismo, mientras que los fuertemente tipificados no.

## 21. Explique el significado de la concurrencia.

“La concurrencia es la propiedad que distingue un objeto activo de uno que no está activo”

Mientras la POO se centra en la abstracción de datos, encapsulamiento y herencia, la concurrencia se centra en la abstracción de procesos y la sincronización. El objeto es un concepto que unifica estos dos puntos de vista distintos: cada objeto (extraído de una abstracción del mundo real) puede representar un hilo separado de control (una abstracción de un proceso). Tales objetos se llaman activos. En un sistema basado en diseño orientado a objetos, se puede conceptualizar el mundo como un conjunto de objetos cooperativos, algunos de los cuales son activos y sirven así como centros de actividad independiente.

Concurrencia permite que diferentes objetos actúen al mismo tiempo, usando distintos threads de control.

## 22. Explique el significado de la persistencia.

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

La persistencia conserva el estado de un objeto en el tiempo y en el espacio.

La persistencia abarca algo más que la mera duración de los datos. En las BD orientadas a objetos, no sólo persiste el

estado de un objeto, sino que su clase debe trascender también a cualquier programa individual, de forma que todos

**los programas interpreten de la misma manera el estado almacenado.**



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

Las características esenciales se denotan **por medio de la abstracción**.

**24. ¿A qué denominamos un objeto cliente?**

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

**25. ¿A qué denominamos un objeto servidor?**

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

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

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

**27. ¿Qué establece un contrato entre objetos?**

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

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

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

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

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

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

Un invariante es una condición booleana cuyo valor de verdad debe mantenerse, si no se mantiene, se rompe el contrato asociado con una abstracción.



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

Se deben definir **pre-condiciones y póst-condiciones**.

### 32. ¿Qué es una pre-condición?

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

### 33. ¿Qué es una post-condición?

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

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

### 34. ¿A qué se denomina excepción?

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

### 35. ¿A qué se denomina mensaje?

**Es la operación que se realiza sobre un objeto, básicamente es una orden que se envía a un objeto para indicarle que realice alguna acción, llamado "pase de mensajes"**. Logrando así que el objeto que envía la petición se denomina emisor y el objeto que recibe la petición se denomina receptor.

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

**Si, la abstracción se centra en el comportamiento observable de un objeto, mientras el encapsulamiento se centra en la implementación que da lugar a ese comportamiento**. El encapsulamiento se consigue a menudo mediante la ocultación de información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales.

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

**Interfaz**

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

## Implementación

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

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

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

La cohesión y que los módulos estén débilmente acoplados.

41. ¿Para qué se utiliza la jerarquía?

Para ordenar abstracciones.

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

Tipo

43. ¿Las clases implementan tipos?

Sí

44. ¿Los tipos implementan clases?

No

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

Lenguajes Tipados

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

Débilmente tipados

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

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

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

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

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

No. Son conceptos independientes. Un lenguaje puede tener comprobación estricta de tipos y tipos estáticos, puede

tener comprobación estricta de tipos pero soportar enlace dinámico, o no tener tipos y admitir la ligadura dinámica.

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

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

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

#### 51. ¿A qué se denomina **concurrencia pesada**?

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

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

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

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

**Si**, por supuesto. En eso se basa.

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

Se denomina **Persistencia**.

#### 55. ¿Qué cosas se persisten?

Persisten entre los objetos de:

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

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

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

Una cosa tangible, o visible, que se puede tocar y comprender intelectualmente. Algo que dirige una acción o pensamiento.

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

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

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

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

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

Las tres cosas que debe tener un objeto son:

- Estado.
- Comportamiento.
- Identidad.

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

Como sinónimo de objeto se puede utilizar la palabra **Instancia**.

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

Como sinónimo se utiliza **Objeto**.

## 62. ¿Cómo definiría el estado de un objeto?

El estado de un objeto abarca todas las propiedades del mismo (normalmente estáticas) más los valores actuales de cada una de esas propiedades (normalmente dinámicas)

todo objeto tiene un estado que implica una cierta cantidad del espacio, ya sea en el mundo físico o en la memoria de la computadora.

### **63. ¿A qué definimos propiedad de un objeto?**

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

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

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

Los valores simples son:

- Intemporales.
- Inmutables.
- No instanciadas.

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

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

### **65. ¿Cómo definiría el comportamiento de un objeto?**

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

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

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

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

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

**Sí.**

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

Sí.

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

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

**70. ¿Cuáles son las tres operaciones más comunes?**

\***Modificador**

\***Selector**

\***Iterador**.

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

\***Constructor**

\***Destructor**

**72. ¿Qué tipo de operación es el modificador?**

Una operación que **altera el estado de un objeto**

**73. ¿Qué tipo de operación es el selector?**

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

**74. ¿Qué tipo de operación es el iterador?**

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

**75. ¿Qué tipo de operación es el constructor?**

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

**76. ¿Qué tipo de operación es el destructor?**

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

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

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

#### **78. ¿Todos los métodos son operaciones?**

**Sí, todos los métodos son operaciones.**

#### **79. ¿Todas las operaciones son métodos?**

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

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

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

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

Papeles o roles.

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

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

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

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

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

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



eventos y al tiempo es tan penetrante que se puede caracterizar mejor formalmente el comportamiento de tales objetos en términos de una máquina de estados finitos equivalentes.

#### 85. ¿Qué es la identidad de un objeto?

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

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

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

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

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

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

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

#### 89. ¿Qué significa realizar una **clonación superficial**?

Significa que copia el objeto y comparte su estado.

#### 90. ¿Qué significa realizar una **clonación profunda**?

Significa que copia el objeto, así como su estado y así sucesivamente.

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

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

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

En lenguajes como Smalltalk, un objeto se destruye automáticamente como parte de la recolección de basura cuando todas las referencias a él se han perdido, en lenguajes sin recolección de basura, como C++, un objeto sigue existiendo y consume espacio incluso si todas las referencias a él han desaparecido. Los objetos en pila son destruidos de manera implícita siempre que el control sale del bloque en el que se declaró el objeto. Los objetos creados con New, deben ser destruidos de manera explícita con el operador delete.

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

Los tipos de relaciones que existen entre los objetos son:

- **Enlaces**: El concepto deriva de Rumbaugh, que lo define como una conexión física o conceptual entre objetos. Un enlace **denota la asociación específica por la cual un objeto utiliza los servicios de otro objeto.**

- **Agregación:** Denota una jerarquía todo/parte, con la capacidad de ir desde el todo hasta sus partes. La agregación es un tipo de especializado de asociación.

#### 94. ¿Cómo podemos definir al enlace entre objetos?

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

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

El paso de mensajes puede ser **unidireccional**, aunque, también puede ser bidireccional.

#### 96. ¿Qué es un mensaje **unidireccional**?

Un mensaje unidireccional es cuando un objeto activo invoca a un objeto o agente que actúe como pasivo.

#### 97. ¿Qué es un mensaje **bidireccional**?

Un mensaje bidireccional es cuando un objeto activo invoca al pasivo y el pasivo le puede devolver datos al objeto activo.

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

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

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

Un objeto puede desempeñar tres papeles o roles:

- **Actor.**
- **Servidor.**
- **Agente.**

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

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

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

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

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

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

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

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

\* El objeto servidor es global para el cliente.

\*El objeto servidor es un parámetro de alguna operación del cliente.

\* El objeto servidor es parte del objeto cliente.

\*El objeto servidor es un objeto declarado localmente en alguna operación del cliente.

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

• **¿Cuáles son las posibles formas de sincronización?**

Secuencial, vigilado y síncrono.

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

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

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

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

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

El funcionamiento del objeto pasivo está garantizado por la utilización de múltiples hilos de control. El servidor

garantiza la exclusión mutua. Hay varios hilos pero ahora el servidor pone el orden y no el cliente como en el caso ante

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

De igual a igual.

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

Es meramente jerárquica.

**112. ¿Qué tipo de jerarquía denota la agregación?**

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

**113. ¿Qué otros nombres recibe el “todo” en una relación de agregación?**

Recibe también el nombre de agregado o contenedor.

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

Si, por supuesto.

**115. ¿Qué tipos de agregación existen?**

Existen dos tipos de agregaciones:

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

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

Cuando un objeto no existe sin el otro.

**117. ¿Qué es una clase?**

Una **clase** es un conjunto de objetos que comparten una estructura común y un comportamiento común.

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

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

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

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

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

Se puede dividir en tres partes:

**Public:** (publica) accesible a todos los clientes.

**Protected:** (protegida) accesible solo a la propia clase, subclases y sus clases amigas.

**Private:** (privada) accesible solo a la propia clase y sus clases amigas.

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

Existen tres tipos:

**Generalización/especialización:** Denota un “es-un” (por ejemplo: una rosa es un tipo de flor)

**Todo/parte:** Denota una relación “parte-de” (Por ejemplo: un pétalo es un tipo de flor, es una parte de ella)

**Asociación:** Denota alguna dependencia semántica entre clases de otro modo independiente (Ejemplo: un picaflor de las flores)

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

Las relaciones son:

Asociación.

Herencia.

Agregación.

Uso.

Instanciación/ejemplares.

Metaclasses.

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

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

#### 124. ¿Qué significa la cardinalidad en una relación?

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

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

Puede existir:

Uno a uno.

Uno a muchos.

Muchos a muchos.

#### 126. ¿Qué es la herencia?

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

#### 127. ¿Cuántos tipos de herencia existen?

Existen dos tipos de herencia:

- Herencia simple.
- Herencia múltiple.

#### 128. ¿A qué se denomina herencia simple?

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

#### 129. ¿A qué se denomina herencia múltiple?

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

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

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

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

**Clase concreta.**

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

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

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

**Superclase** o clase base.

### **134. ¿Qué es el polimorfismo?**

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

- **¿Cómo se denomina cuando una clase posee métodos que comparten el nombre y se diferencian por su firma?**

Se denomina **Sobrecarga**.

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

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

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

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

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

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

\*Con contención Física (se subdivide en valor y referencia)

\* Sin contención Física. (el ciclo de vida de los elementos son independientes uno del otro)



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

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

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

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

### **141. ¿Qué es una relación de uso?**

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

#### 142. ¿Qué es la instanciación?

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

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

**Sí.**

#### 144. ¿Qué es una metaclass?

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

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

\*Acoplamiento

\*Cohesión

\*Suficiencia

\*Compleción

\*Ser primitivo

#### 146. ¿Qué es el acoplamiento?

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

#### 147. ¿Qué es la cohesión?

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

#### 148. ¿Qué es la suficiencia?

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

#### 149. ¿Qué es la completación?

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

#### **150. ¿Qué significa ser primitivo?**

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

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

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

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

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

El paso de mensajes debe adoptar las siguientes formas:

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

#### **153. ¿Qué características posee un mensaje síncrono?**

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

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

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

#### **155. ¿Qué características posee un mensaje de intervalo?**

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

#### **156. ¿Qué características posee un mensaje Asíncrono?**

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

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

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

### **158. ¿Qué expresa la ley de Demeter?**

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

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

La consecuencia inmediata de aplicar ésta ley es la creación de clases débilmente acopladas, cuyos secretos de implantación están encapsulados. Tales clases están claramente libres de sobrecargas, lo que significa que para comprender el significado de una clase no es necesario comprender los detalles de muchas otras clases.

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

Existen cuatro formas fundamentales:

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

### **161. ¿Para qué sirve clasificar a los objetos?**

La clasificación es el medio por el que ordenamos el conocimiento. En el diseño orientado a objetos, el reconocimiento de la solicitud entre las cosas nos permite exponer lo que tienen en común en abstracciones clave y mecanismo y eventualmente nos lleva a arquitecturas más pequeñas y simples. No existe un camino trillado hacia la clasificación.

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

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

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

El rol es trascendental, ya que el problema central de observación es sin duda el observador, porque debe asimilar la información derivada de sus observaciones y

después sacar conclusiones acerca de sus construcciones hipotéticas. Puede hacer inferencias por completo erróneas. Por el contrario, si el observador es por completo objetivo y no conoce el tema de la observación puede que lo observado no sea lo adecuado. La observación exige un conocimiento competente de lo observado y de su significado.

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

Las aproximaciones generales a la clasificación son tres:

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

#### **165. ¿Qué es la categorización clásica?**

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

#### **166. ¿Qué es el agrupamiento conceptual?**

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

Resumiendo, se definen pautas descriptivas. Se desarrolla una estructura conceptual. Se agrupan todas aquellas que

respondan a la descripción establecida.

#### **167. ¿Qué es la teoría de prototipos?**

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

#### **168. ¿Qué es una abstracción clave?**

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

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

\*Los objetos deben denominarse con nombres propios, mientras las clases tienen nombres comunes.

\* Operaciones modificadoras deben denominarse con verbos activos, mientras las operaciones de selección usan preguntas o nombres con el verbo ser.

### 169. ¿Qué son los mecanismos?

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

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

## UNIDAD 2

### 170. ¿Qué es un Campo de una clase?

Es una propiedad o atributo.

### 171. ¿Qué es un método de una clase?

Es un procedimiento.

### 172. ¿A qué denominamos sobrecarga?

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

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

### 173. ¿Qué es una propiedad de una clase?

Los atributos que posee.

### 174. ¿Qué tipos de propiedades existen?

Existen cuatro tipos de propiedades:

- Solo Lectura. (ReadOnly Property Edad) - { get; }
- Solo Escritura. (WriteOnly Property Edad) - { set; }
- Lectura/Escritura. (Por defecto) - { get; set; }
- Propiedades con Argumentos.

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

Se engloban en tres ámbitos:

- Public. - public.
- Private. - private.
- Friends. - protected.

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

Las características son:

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

### 177. ¿Para qué se utilizan los constructores?

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

```
Auto(){
```

```
}
```

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

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

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

No, no se utiliza un contador de referencia.



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

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

**181. ¿Qué son los sucesos?**

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

**182. ¿Qué se utiliza para declarar un suceso?**

Se utiliza: **Event** (nombre)

**183. ¿Cómo se logra que ocurra un suceso?**

Se logra a partir de: *RaiseEvent* (nombre)

**184. ¿Cómo se pueden atrapar los sucesos?**

Los sucesos se atrapan con: *WithEvents*

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

Utilizar Addhandler permite decidir en tiempo de ejecución que rutina deberá dar servicio a un determinado suceso. AddHandler acepta 2 argumentos:

- El suceso que voy a redirigir a un procedimiento de la aplicación (en el formato objeto.nombresuceso)
- La dirección de la rutina que controla el suceso (en el formato AddressOfnombrerutina).

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

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

**187. ¿Qué características poseen los campos compartidos?**

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

Los campos compartidos también resultan de utilidad cuando todas las instancias de una clase compiten entre sí por un número limitado de recursos.

**188. ¿Qué características poseen los métodos compartidos?**

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

### 189. ¿Qué características poseen los sucesos compartidos?

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

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

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

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

Existen cinco ámbitos de clases anidadas:

- **Público:** El ámbito público hace que una clase, o cualquiera de sus miembros, sea visible fuera del ensamble actual si el proyecto es un proyecto de “biblioteca”
- **Privado:** El ámbito privado convierte, justamente, en privada a una clase y utilizable únicamente dentro de su contenedor. Este contenedor suele ser normalmente la aplicación actual salvo en el caso de las clases anidadas (una clase anidada privada sólo se puede utilizar dentro de su clase contenedora). Un miembro privado es utilizable únicamente dentro de la clase en la que se ha definido y esto incluye cualquier clase anidada en el mismo contenedor.
- **Friend:** El ámbito friend combina las características de las palabras clave `Friend` y `Protected` y, por tanto, define un miembro o clase anidada que es visible en todo el ensamblado y en todas las clases heredadas.
- **Protected:** El ámbito protected hace que un miembro o una clase anidada sea visible dentro de la clase actual, así como en todas las clases derivadas de la clase actual. Los miembros protected son miembros privados que también son heredados por las clases derivadas
- **Protected Friend:** Este ámbito combina las características de las palabras clave `Friend` y `Protected` y, por tanto, define un miembro o clase anidada que es visible en todo el ensamblado y en todas las clases heredadas.

### 192. ¿Qué cosas se pueden heredar?

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

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

El polimorfismo se utiliza para la reutilización del código y mayor eficiencia y legibilidad del mismo. En la clase base colocar antes de la firma del método `Overrideable`. En la clase derivada colocar `Overrides` (Redefinir)

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

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

**195. ¿Qué representa la clase `this`?**

Representa el contexto actual de la clase en ejecución.

**196. ¿Qué clase representa a la clase base?**

Representa la clase `MyBase`

**197. ¿Para qué se usa una clase abstracta?**

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

**198. ¿Para qué se usa una clase sellada?**

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

**199. ¿Qué es la sobreescritura?**

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

**200. ¿Qué elementos se pueden sobrecribir?**

Se puede sobrecribir los métodos.

**201. ¿A qué se denomina sombreado de métodos?**

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

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

El sombreado se caracteriza porque sobreescribe y oculta los otros métodos de la sobrecarga. Un miembro de la clase derivada, marcado con la palabra clave *Shadow*, oculta todos los miembros de la clase base que tengan el mismo nombre, con independencia de sus firmas. Un miembro de la clase derivada, marcado con la palabra clave *Overloads*, sombreadá únicamente al miembro contenido en la clase base que tenga el mismo nombre y firma de argumentos.

### **UNIDAD 3**

#### **203. ¿Qué es un Framework?**

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

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

Características principales:

- 1- Normalizar: homogeneizar para usarlo en forma común y colocarlo en el framework
- 2- Estandarizar: generar documentación
- 3- Reutilizar experiencia

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

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

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

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

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

\*De Caja Negra

\*De Caja Blanca

## 207. ¿Qué es un Framework de Caja Blanca?

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

## 208. ¿Qué es un Framework de Caja Negra?

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

Los frameworks de caja negra producen instancias usando escrituras o scripts de configuración. Después de la

configuración, una herramienta automática de instanciación crea las clases y el código de fuente (Source Code). La

caja negra del framework no requiere que el usuario conozca los detalles internos del mismo. Para su instanciación no

es preciso conocer la forma en que están contruidos. El desarrollador con personaliza los puntos calientes mediante la instanciación con parámetros actuales y la composición.

## 209. ¿Qué ventajas posee utilizar un Framework?

- Permite abstraerse de problemas resueltos con amplia experiencia previa.
- En el caso de .NET el programador no piensa en arquitectura sino en programas.
- Permite aprovechar todas las ventajas de la estandarización.

\*Las aplicaciones crecen y necesitan mantenimiento. Los frameworks facilitan la separación de la presentación y la lógica además de mantener una sintaxis coherente.

## 210. ¿Qué problemas resuelve .NET Framework?

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

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

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

## 212. ¿Qué es el MSIL?

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

## 213. ¿Qué es el CTS?

**Define un conjunto común de “tipos” orientado a objetos**, los cuales son tres:

- **Todo lenguaje de programación debe implementar los tipos definidos por el CTS.**
- **Todo tipo hereda directa o indirectamente del tipo OBJECT.**
- **Todo tipo de VALOR y de REFERENCIA.**

## 214. ¿Qué es el CLS?

La interoperabilidad entre lenguajes es la posibilidad de que el código interactúe con código escrito en un lenguaje de programación diferente. La interoperabilidad entre lenguajes puede ayudar a maximizar la reutilización de código y, por tanto, puede mejorar la eficacia del proceso de programación. Dado que los desarrolladores utilizan una gran variedad de herramientas y tecnologías, cada una de las cuales podría admitir distintos

tipos y características, desde tiempo atrás ha sido complicado garantizar la interoperabilidad entre lenguajes. No obstante, los compiladores y las herramientas de lenguaje dirigidos a Common Language Runtime se benefician de la compatibilidad que integra el motor en tiempo de ejecución para la interoperabilidad entre lenguajes. Para garantizar que el código administrado será accesible para los desarrolladores que empleen otros lenguajes de programación, .NET Framework proporciona **Common Language Specification (CLS)**, que describe un conjunto fundamental de características de lenguaje y define reglas sobre cómo usar dichas características.

## 215. ¿Qué es una excepción?

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

## 216. ¿Qué se coloca en el bloque “Catch”?

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

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

*Class MiExcepcionPersonalizaException Inherits System.ApplicationException*

*class excepcionMia : Exception*

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

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

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

El de mayor jerarquía es: **Exception**.

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

Se encuentra en: **System.Exception**

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

Las clases genéricas más importantes son:

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



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

Se utilizan:

*Try.*  
*Catch.*  
*End Try.*

*Try.*

*Catch.*

*When.*

*Finally*

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

Try/ End Try

try { }

**224. ¿Qué se coloca en el bloque “Catch”?**

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

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

**DivideByzeroException**

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

System.DLLNotFoundException

**227. ¿Qué colocaría dentro de una clausula “Catch” para especificar una condición adicional que el bloque “Catch” deberá evaluar como verdadero para que sea seleccionada?**

**When**

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

**Dentro del bloque Finally porque siempre se ejecuta.**

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

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

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

```
if(age<7 && number > 10)
throw new ArithmeticException("Number unsuitable for age group");
```

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

**Class MiExcepcionPersonaliza : Exception**

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

```
Try{
}catch( ExceptionMia ex){
}
```

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

En el Managed Heap, o montón administrado.

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

**Finalize()** y **Dispose()**.

**235. ¿De dónde heredan las clases el método Finalize?**

System.Object

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

Finalize()

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

**GC.Collect()**

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

GC.WaitForPendingFinalizers()

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

Finalize()

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

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

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

El código para limpieza/ liberación de recursos.

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

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

**243. ¿A qué se denomina “Resurrección de Objetos”?**

Un objeto que haya sido finalizado puede almacenar una referencia de sí mismo en una variable definida fuera de la clase para que esta nueva referencia mantenga vivo al objeto, a esto llamamos Resurrección de Objetos

**244. ¿A qué se denomina “Generación” en el contexto de la recolección de elementos no utilizados?**

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

**245. ¿Qué valores puede adoptar la “Generación” de un objeto?**

Hay sólo tres valores distintos de generación. Tomando en cuenta que el valor de la generación de un objeto que nunca ha sufrido una recolección de elementos no utilizados es 0;

- Si el objeto sobrevive a una recolección de elementos no utilizados su generación vale 1.
- Si sobrevive a una segunda recolección de elementos no utilizados, su generación valdrá 2.

- Cualquier posterior recolección de elementos no utilizados hace que el contador de generación siga valiendo 2 (o se destruye el objeto)

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

`GC.Collect(0)`

Como parámetro le paso el número de generación que quiero verificar.

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

Se obtiene desde: `GC.GetGeneration()`

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

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

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

Se utiliza: `GC.SuppressFinalize()`

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

Se solicita por medio de: `GC.SuppressFinalize()`

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

Se solicita al: `GC.WaitForPendingFinalizers()`

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

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

## **UNIDAD IV**

**253. ¿Qué son y para que se usan los tipos genéricos?**

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

**254. Enumere las ventajas de utilizar genéricos.**

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

### 255. ¿A qué elementos se le pueden aplicar genéricos?

A las variables y listas se les puede aplicar Genéricos,

### 256. ¿Qué es LinQ to Object?

Es un lenguaje Integrado de Consulta que sirve para poder hacer consultas sobre Colecciones de Objetos, Bases de Datos y Archivos XML.

### 257. ¿Qué es una expresión Lambda?

Son funciones anónimas que se usan frecuentemente para crear delegados en LINQ. Son simples, sin ninguna declaración, es decir, sin modificadores de acceso, la declaración de valor de retorno y nombre.

Es una especie de taquigrafía que permite abreviar el método en el mismo lugar que se usará. Nos ahorra tiempo, espacio y además, es fácil de revisar el código.

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

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

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

---

## UNIDAD V

### 259. ¿Para qué se utilizan las interfaces?

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

### 260. ¿Cómo se implementa una interfaz?

*Public Interface xx ... End Interface*

```
public interface Ifranco {  
  
}
```

#### 261. ¿Se pueden heredar las interfaces?

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

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

Si, se puede implementar.

#### 263. ¿Para qué se utiliza la interfaz IComparable?

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

#### 264. ¿Para qué se utiliza la interfaz IComparer?

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

#### 265. ¿Para qué se utiliza la interfaz ICloneable?

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

#### 266. ¿Para qué se utiliza la interfaz IEnumerable?

La interfaz **IEnumerable** se utiliza para enumerar elementos de una clase desde afuera de la misma, entregando una lista de elementos y no la colección completa. Obliga a implementar `GetEnumerator()` que devuelve un `Enumerator`. Recupera todos los elementos de un enumerador. Esta interfaz deberá devolver un objeto que permita el empleo de la interfaz `Ienumerator`.

#### 267. ¿Para qué se utiliza la interfaz IEnumerator?

Para crear un numerador personalizado. Posee los siguientes métodos

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

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

## 268. ¿Qué es un delegado?

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

## 269. ¿A qué elementos se les puede delegar?

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

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

## 270. ¿Qué se puede delegar?

\* Eventos

\* Funciones

## 271. ¿Cómo construiría un delegado?

```
Dim deleg as SubUnArg  
deleg= New SubUnArg (AdressOf MostrarMsg)
```

```
Delegate Sub SubUnArgr (ByVal msg as String)
```

```
public delegate void MiDelegado(int i);
```

## 272. ¿Cómo implementaría un procedimiento de devolución de llamadas?

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

**273. ¿Para qué sirve la multidifusión de delegados?**

*Para enviar una llamada a más de un procedimiento.*

**UNIDAD 6**

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

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

**275. ¿Qué significa pasar información batch?**

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

**276. ¿Qué significa pasar información on line?**

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

**277. ¿Qué es un protocolo?**

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

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

**TCP/IP**

**279. ¿Qué protocolo usa internet?**

**IP**

**280. ¿Qué hace el protocolo IP?**



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

### **281. ¿Qué ventajas tiene distribuir procesos?**

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

### **282. ¿Qué ventajas tiene distribuir almacenamientos?**

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

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

### **283. ¿Qué es un socket?**

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

### **284. ¿Qué característica posee un socket sincrónico?**

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

### **285. ¿Qué característica posee un socket asincrónico?**

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

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

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

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

### 288. ¿Qué puertos posee una PC?

USB – RJ45 – paralelo (old) – serie (old) – jack 3.5.

### 289. ¿Cómo funciona el puerto paralelo?

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

### 300. ¿Cómo funciona el puerto serie?

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

### 301. ¿Cómo funciona el puerto USB?

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

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

### 302. ¿Qué es la domótica?

La **domótica** es la integración de tecnología en el diseño inteligente o automatizado de un recinto (Casa, Apartamento, Casas campestres, fincas, lugar de trabajo etc...) con funciones de información, entretenimiento, gestión energética, seguridad y búsqueda de soluciones a la medida y aplicaciones según sus necesidades. Entonces, **se trata de una ciencia que estudia la aplicación de la informática y las comunicaciones al hogar.**