GC-F-004v.08

Sesión en línea

Agosto 10/2023

Orientaciones

Desarrollo evidencias Guía
Aprendizaje 4 (GA4)
Diagrama de clases del proyecto de software
GA4-220501095-AA2-EV04

GA4-220501095-AA2-EV04

GA4-220501095-AA2-EV04 - Diagrama de clases del proyecto de software

Introducción

Para la construcción de software se hace evidente la importancia de la utilización de modelos ya que representan el aspecto y la dirección de una necesidad. Esa “necesidad” puede existir, estar en un estado de desarrollo o en un estado de planeación. Los diseñadores del modelo deben realizar un levante de los requerimientos del producto y dichos requerimientos pueden incluir áreas tales como funcionalidad, performance y confiabilidad. El modelo se divide en un número de vistas, cada una de las cuales describe un aspecto específico del producto o sistema en construcción. Un modelo permite documentar la estructura y conducta de un sistema antes de que sea codificado.

GA4-220501095-AA2-EV04



GA4-220501095-AA2-EV04

Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. En un diagrama de clases se pueden distinguir principalmente dos elementos: clases y sus relaciones, (Flores Cueto & Bertolotti Zuñiga, 2013).

GA4-220501095-AA2-EV04

Diagrama de clases

El objetivo de un diagrama de clase es representar las clases que conforman el modelo de un determinado sistema, el diagrama de clase se construye y se refina durante las fases de análisis y diseño, y se toma como referencia en la implementación del sistema.

Hay tres perspectivas diferentes en las cuales se pueden utilizar los diagramas de clase:

- **Conceptual**
- Especificación
- Implementación



El diagrama de clase representa los conceptos en el dominio del problema que se está estudiando.

GA4-220501095-AA2-EV04

Diagrama de clases

El objetivo de un diagrama de clase es representar las clases que conforman el modelo de un determinado sistema, el diagrama de clase se construye y se refina durante las fases de análisis y diseño, y se toma como referencia en la implementación del sistema.

Hay tres perspectivas diferentes en las cuales se pueden utilizar los diagramas de clase:

- Conceptual
- **Especificación**
- Implementación



El diagrama de clase refleja las interfaces de las clases, pero no su implementación. Aquí las clases aparecen más cercanas a los tipos de datos, ya que un tipo representa una interfaz que puede tener muchas implementaciones diferentes.

GA4-220501095-AA2-EV04

Diagrama de clases

El objetivo de un diagrama de clase es representar las clases que conforman el modelo de un determinado sistema, el diagrama de clase se construye y se refina durante las fases de análisis y diseño, y se toma como referencia en la implementación del sistema.

Hay tres perspectivas diferentes en las cuales se pueden utilizar los diagramas de clase:

- Conceptual
- Especificación
- **Implementación**



Esta vista representa las clases tal cual aparecen en el entorno de implementación.

GA4-220501095-AA2-EV04

Diagrama de clases

Los elementos que componen un diagrama de clase son:

- **Clase**
- Atributos
- Métodos



Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una casa, un auto, una cuenta corriente, etc.).

- Clase
- **Atributos**
- Métodos



Los atributos o características de una clase.

GA4-220501095-AA2-EV04

Diagrama de clases

Los elementos que componen un diagrama de clase son:

- Clase
- Atributos
- **Métodos**



Los métodos u operaciones de una clase son la forma en cómo ésta interactúa con los datos.

En UML, una clase es representada por un rectángulo que posee tres divisiones:

Representación de una clase en UML

Nombre de la Clase
- Atributos
+Operaciones o Métodos ()

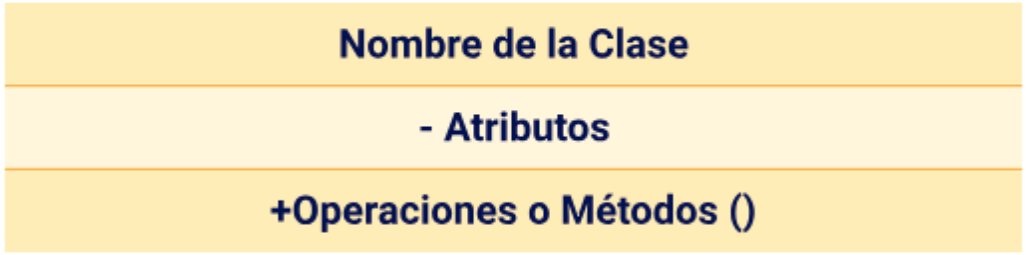
Fuente: SENA, 2021

GA4-220501095-AA2-EV04

Diagrama de clases

En UML, una clase es representada por un rectángulo que posee tres divisiones:

Representación de una clase en UML



Fuente: SENA, 2021

En donde:

- El rectángulo superior contiene el nombre de la clase.
- El rectángulo intermedio contiene los atributos (o variables de instancia) que caracterizan a la clase.
- El rectángulo inferior contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno.

Los atributos y las clases tienen características que definen el grado de comunicación y visibilidad de ellos con el entorno a lo que llamamos controles de acceso.

GA4-220501095-AA2-EV04

Diagrama de clases – Control de acceso

El control de acceso o la visibilidad define la accesibilidad para los atributos o métodos de la clase. Los atributos y métodos pueden ser privados, protegidos o públicos y se representan con cada uno de los símbolos que muestra la siguiente tabla:

Tabla 1. Símbolos de modificadores de acceso

Público	+
Privado	-
Protegido	#

Fuente: SENA, 2021

GA4-220501095-AA2-EV04

Diagrama de clases – Control de acceso

El control de acceso o la visibilidad define la accesibilidad para los atributos o métodos de la clase. Los atributos y métodos pueden ser privados, protegidos o públicos y se representan con cada uno de los símbolos que muestra la siguiente tabla:

Público	 Se define un atributo o método público cuando se quiere que sean vistos o accesibles por todas las clases
Privado	
Protegido	

GA4-220501095-AA2-EV04

Diagrama de clases – Control de acceso

El control de acceso o la visibilidad define la accesibilidad para los atributos o métodos de la clase. Los atributos y métodos pueden ser privados, protegidos o públicos y se representan con cada uno de los símbolos que muestra la siguiente tabla:

Público	
Privado	 Se define un atributo o método privado cuando se quiere que sean visibles o accesibles únicamente por la misma clase en que están.
Protegido	

GA4-220501095-AA2-EV04

Diagrama de clases – Control de acceso

El control de acceso o la visibilidad define la accesibilidad para los atributos o métodos de la clase. Los atributos y métodos pueden ser privados, protegidos o públicos y se representan con cada uno de los símbolos que muestra la siguiente tabla:

Público

Privado

Protegido



Se define un atributo o método protegido cuando sólo se quiere tener acceso a él desde la propia clase que lo define y las que heredan de él.

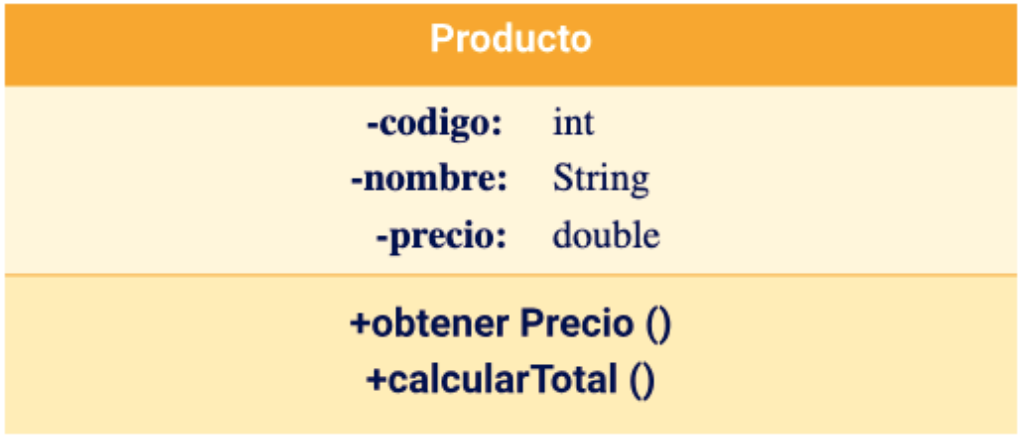
GA4-220501095-AA2-EV04

Diagrama de clases – Ejemplo

Por ejemplo, se puede representar una clase denominada Producto que contiene 3 atributos (código de tipo entero, nombre de tipo String y precio de tipo double) y 2 métodos (obtenerPrecio y calcularTotal), de la siguiente manera:

Se observa que todos los atributos en esta clase son privados porque tienen el símbolo – y los métodos son públicos porque tienen el símbolo +

Representación de la clase Producto en UML



Fuente: SENA, 2021

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Las relaciones existentes entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí, existen diferentes tipos de relaciones

- Asociación (conexión entre clases).
- Dependencia (relación de uso).
- Generalización (relaciones de herencia).

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Asociación (conexión entre clases): Se le conoce como relación de asociación a las conexiones entre clases. Este tipo de relación se representa gráficamente como una línea que une las dos clases y tiene las siguientes características:

1 Nombre de la asociación. (Es obligatorio establecer el nombre de la asociación).

2 Cada clase tiene asociado un rol, el cual se interpreta como el papel que juega dicha clase en el contexto de la asociación. Los roles son opcionales y no se utilizan cuando el nombre de la clase expresa correctamente el rol de esta en el contexto de la relación de asociación.

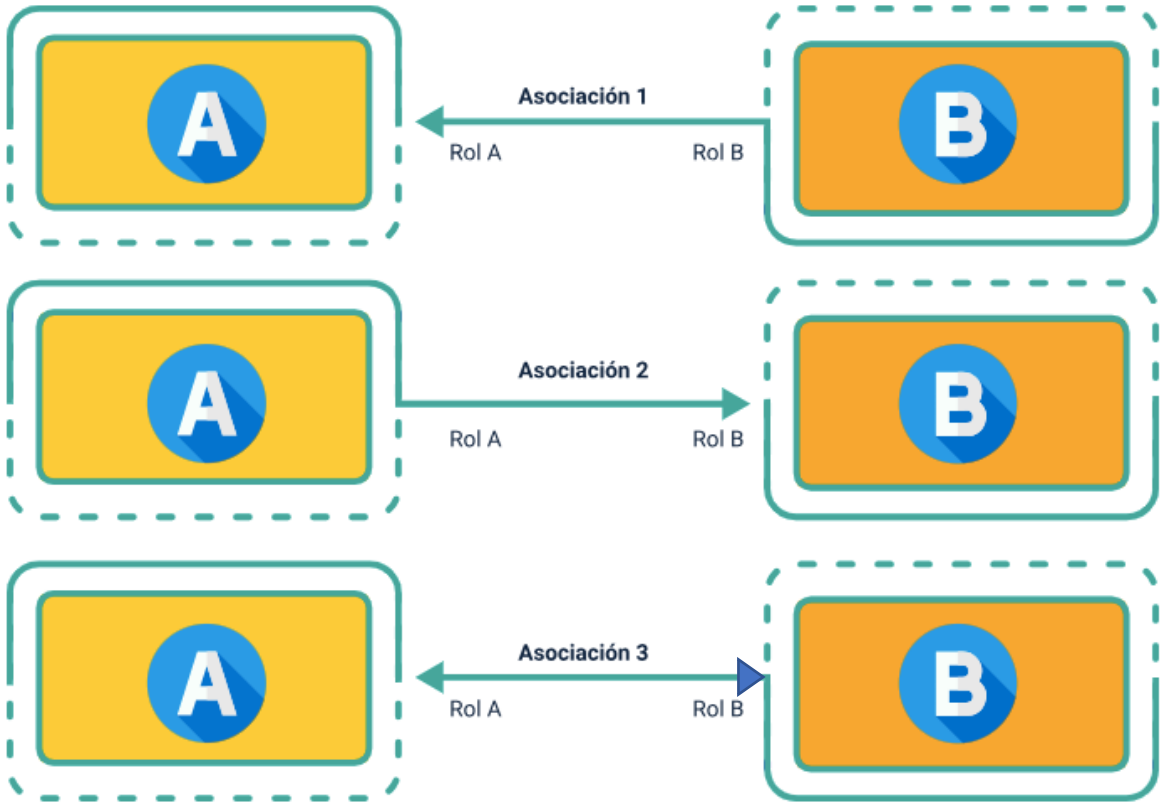
3 Navegabilidad: en las relaciones de asociación se puede establecer si el vínculo es unidireccional o bidireccional. Gráficamente se representan con puntas de flecha.

4 Multiplicidad: representan el número de instancias de la clase que pueden ser partícipes en el proceso de asociación con respecto a una instancia particular de la otra clase vinculada en la relación. (Schmuller, 2001)

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Asociación (conexión entre clases):



Fuente: SENA, 2021

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Asociación (conexión entre clases): En el ejemplo anterior, se pueden observar 3 diferentes ejemplos de asociación entre la clase A y la clase B, en todos los ejemplos la Clase A asume el rolA en el contexto de asociación, mientras que la clase B asume el rolB en el contexto de la relación.

Adicionalmente, se puede observar que los dos primeros ejemplos tienen navegabilidad unidireccional, lo cual representa que la relación se establece en un solo sentido. En el primer caso la Asociación1 vincula la clase B como origen y la clase A como destino, en el segundo caso se vincula con la Asociación2 a la clase A como origen con la clase B como destino y en el último ejemplo el vínculo es bidireccional, es decir en los dos sentidos, desde A hacia B y desde B hacia A.

Una relación de asociación debe tener como mínimo tres elementos: (I) nombre de la asociación, (II) navegabilidad y (III) multiplicidad. Respecto a la multiplicidad esta representa la cantidad numérica de instancias de las clases que participan en la relación, aunque este puede ser cualquier número normalmente se acostumbra a utilizar el valor de 1 o * cuando son varias instancias las participantes.

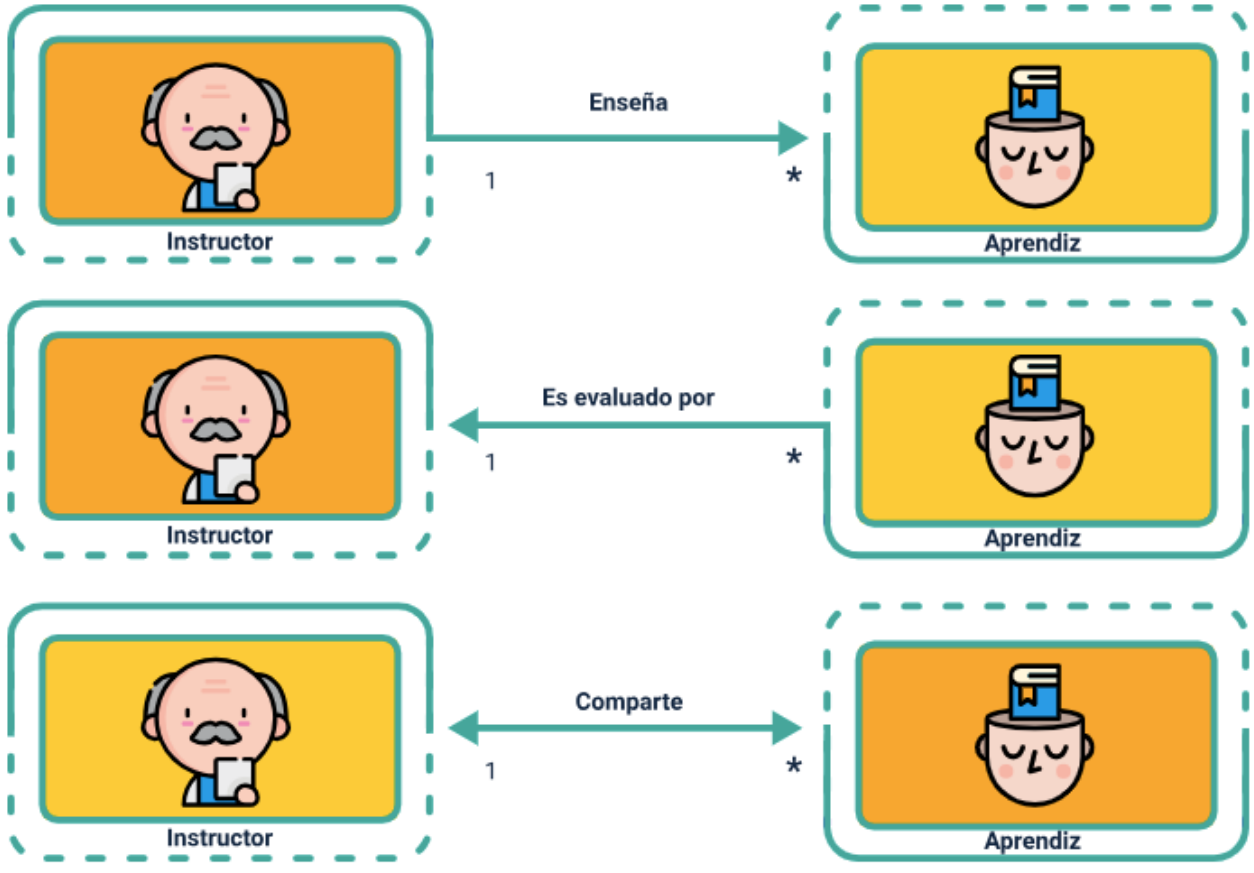
Para facilitar el proceso de entendimiento y lectura se debe tener en cuenta la navegabilidad. Siempre se lee tomando como referencia un ejemplar de la clase desde donde se origina la asociación respecto al valor N de instancias del otro extremo de la asociación, este valor N es la multiplicidad en el otro extremo.

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases: Ejemplo

Asociación (conexión entre clases): En el ejemplo se presentan tres variaciones de relaciones de asociación entre la clase instructor y la clase aprendiz, en todos los tres casos la navegabilidad varía dependiendo de lo que se está representando en la relación por medio de su nombre. En el primer caso es una relación de asociación llamada Enseña con navegabilidad desde Instructor hacia Aprendiz indicando que un instructor enseña a varios aprendices.

En el segundo caso se lee de la siguiente forma: un aprendiz es evaluado por un instructor. En el último caso de esta gráfica se representa una relación bidireccional, es decir, se hacen dos lecturas desde cada extremo, en este ejemplo un instructor comparte con muchos aprendices y un aprendiz comparte con un instructor.



Fuente: SENA, 2021

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Generalización (relaciones de herencia): La herencia es una de las relaciones más comunes en la práctica del paradigma orientado a objetos y como se mencionó en el componente formativo anterior se puede construir relaciones partiendo desde conceptos (clases) generales a conceptos (clases) específicos (especialización) o desde conceptos específicos hacia conceptos generales (generalización), (Schmuller, 2001).

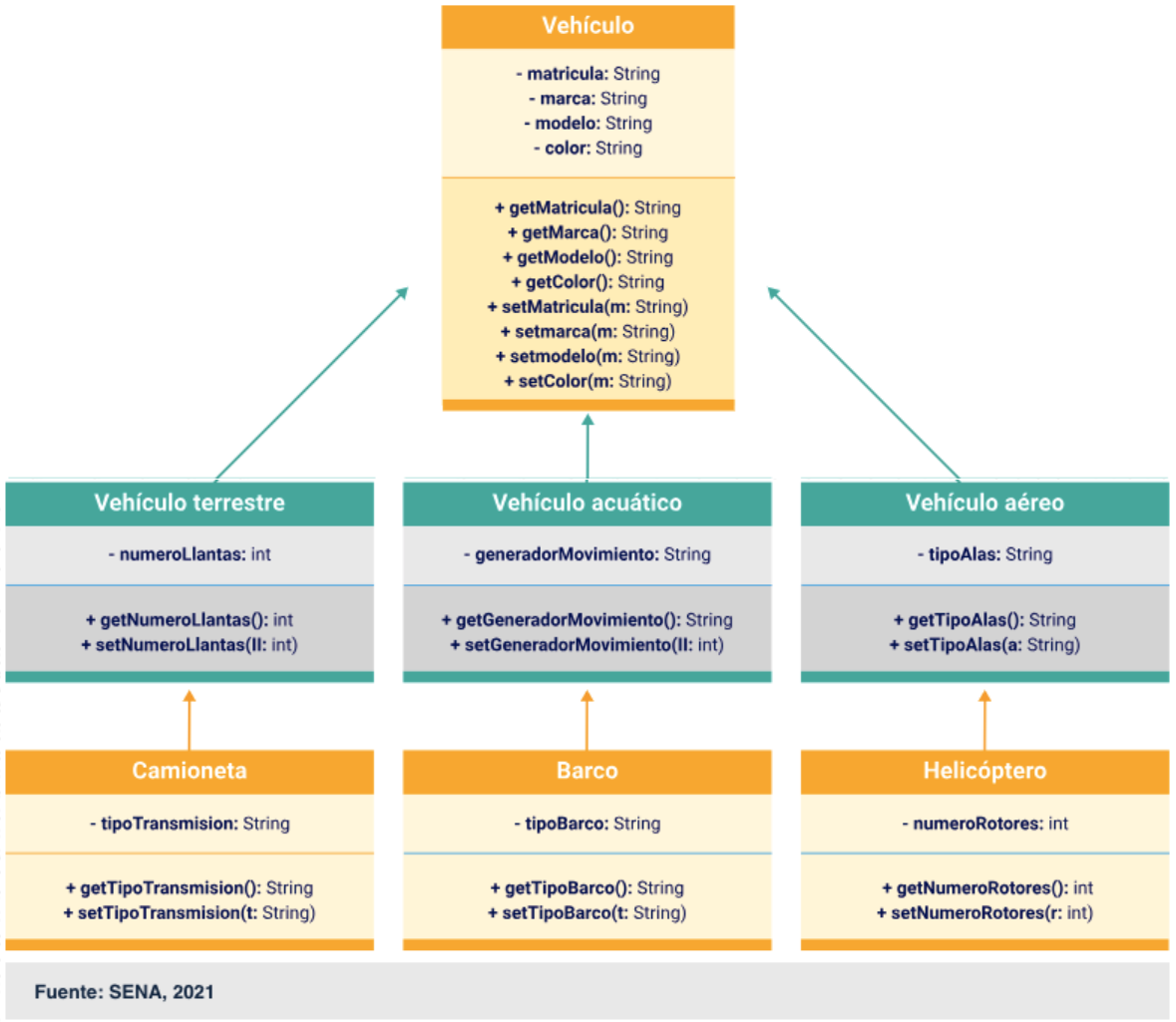
Es un tipo especial de asociación donde se tienen clases principales o superclases (las más generales en la relación) y clases secundarias o subclases (las clases más especializadas), adicionalmente este tipo de relación tiene implícito el nombre es un o es una y también implícitamente es una relación que vincula a un ejemplar de tipo superclase y un ejemplar de tipo subclase, por lo anterior no se colocan nombre ni multiplicidad a este tipo de relación.

En una relación de herencia las subclases heredan las características (atributos) y los comportamientos (métodos) de las superclases.

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Ejemplo de herencia



Fuente: SENA, 2021

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

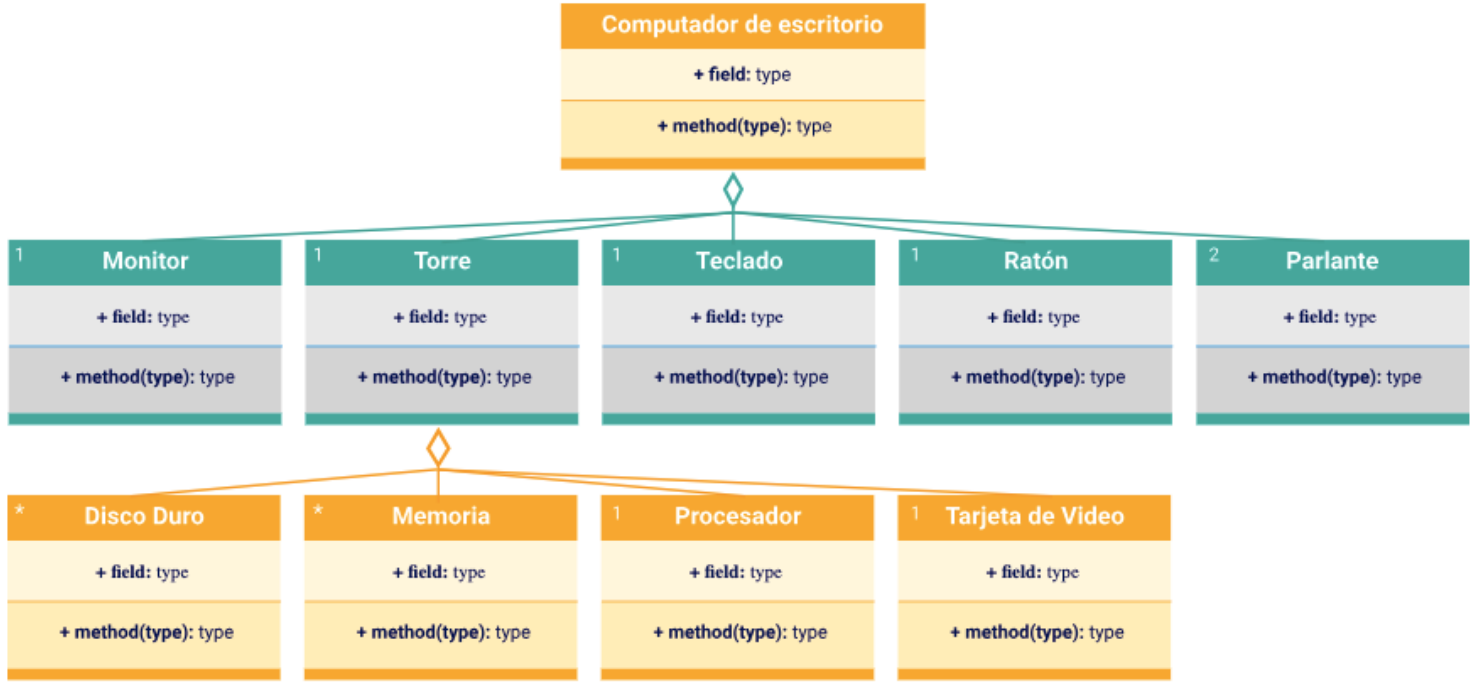
Dependencia (relación de uso): La agregación es un tipo de relación especial, donde se representa que un conjunto de clases conforma un todo, es decir, existen clases agregadas que representan el todo y se constituyen a partir de un conjunto de clases componentes (Schmuller, 2001).

Este tipo de relación se representa con una línea que une la clase agregada junto a sus clases componentes, del lado de la clase agregada tiene un rombo que representa la relación de agregación. Esta relación no tiene nombre explícito ya que se entiende que hay clases agregadas y clases componentes que la forman, pero si es necesario establecer la multiplicidad del lado de los componentes.

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Dependencia (relación de uso): En la figura, se muestra un ejemplo de relación de agregación, como se puede observar la clase Computador de escritorio se compone de un monitor, una torre, un teclado, un ratón y dos parlantes. A su vez la clase Torre se compone de varios discos duros, varias memorias, un procesador y una tarjeta de video. Los elementos que son componentes en este diagrama pueden ser componentes de otras clases, además una instancia de los componentes puede ser reemplazado por otra fácilmente razón por la cual la relación que tienen con la clase agregada no es tan fuerte y por eso es una agregación.



GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

Video tutorial de apoyo: https://youtu.be/7WRYH2ei5Rw

GA4-220501095-AA2-EV04

Diagrama de clases – Relaciones entre clases

- Diseñar 1 solo Diagrama de clases aplicado sobre el proyecto de software
- Aplicar buenas prácticas de diseño orientado a objetos.
- Utilizar una herramienta TIC para hacer diagramas, recomiendo emplear StarUML

Pista clave 1: Estudiar el componente formativo "*Diseño del modelo conceptual bajo el paradigma orientado a objetos*".

Pista clave 2: **Video tutorial de apoyo**: https://youtu.be/7WRYH2ei5Rw

Estructura del documento:

Normas básicas de presentación de un documento escrito, debe contener como mínimo una portada, introducción, objetivos, desarrollo del trabajo y conclusiones.

GA4-220501095-AA2-EV06

Evidencias de producto: GA4-220501095-AA2-EV06 - Taller arquitectura de software

Elaborar un taller sobre arquitectura de software teniendo en cuenta las siguientes preguntas:

- ¿Qué entiende por arquitectura de software? Definición
- ¿Cuál es su función? Definición – **Traten de representar lo que investigan en un grafico**
- ¿Cómo se elabora la arquitectura de software?
- ¿Cómo lograr una buena arquitectura?
- ¿Cuáles son los elementos de diseño de una arquitectura de software?

Pista clave 1: Estudiar el componente formativo "*Diseño de patrones de software*".

Estructura del documento:

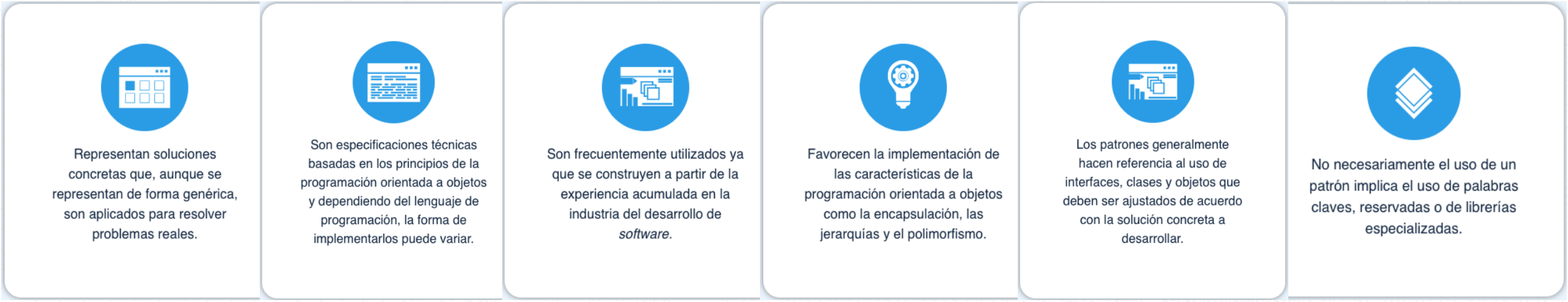
Normas básicas de presentación de **un documento escrito**, debe contener como mínimo una portada, introducción, objetivos, desarrollo del trabajo, conclusiones y referencias

GA4-220501095-AA2-EV05

Evidencia de desempeño: GA4-220501095-AA2-EV05 - Desarrollar la arquitectura de software de acuerdo al patrón de diseño seleccionado

Conceptos sobre patrones de diseño

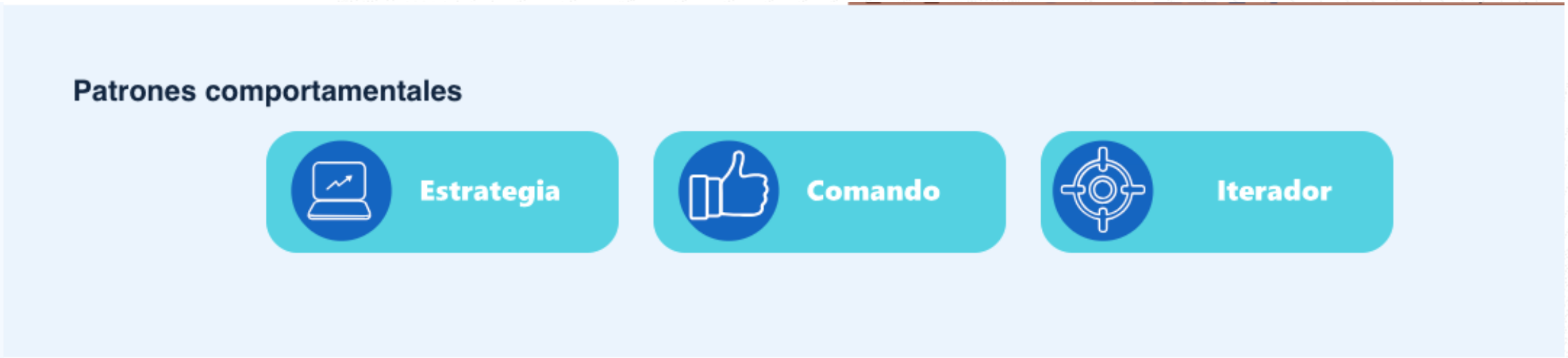
El uso de patrones de diseño en el ámbito de la industria de software, implica el conocimiento y la aplicación de una serie de criterios que aportan calidad al proceso mismo y efectividad del producto logrado. En ese mismo sentido, se favorece la estandarización del código fuente de determinada aplicación, lo cual facilita su monitoreo y mantenimiento. Los patrones de diseño tienen las siguientes características:



GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones comportamentales

Los patrones de diseño comportamentales se centran en definir la manera en que los objetos interactúan entre ellos, por medio de mensajes. Son tres los tipos de patrones comportamentales: estrategia, comando e iterador.



GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones comportamentales

Estrategia: El patrón estrategia permite encapsular un conjunto de algoritmos de manera que puedan ser seleccionados dinámicamente para su ejecución, durante el tiempo de ejecución y de acuerdo con las acciones del cliente. Este patrón es una de las formas en las que se ven reflejadas fácilmente, las características de la programación orientada a objetos, particularmente en lo referente a encapsulamiento y polimorfismo.
Este patrón de diseño es útil cuando una misma funcionalidad puede ser provista usando diferentes mecanismos, algoritmos o estrategias, que serán seleccionadas dependiendo de las acciones realizadas por el cliente, en el momento que está ejecutando el programa.

Ejemplo: suponga que quiere implementar una calculadora, la cual provee un conjunto de operaciones (suma, resta, multiplicación y división) que serán usadas por el cliente según su deseo. Cada una de estas operaciones representan una estrategia diferente y será el cliente quien invocará la ejecución de cada una de ellas dependiendo de su deseo por medio de una interfaz que, usando las propiedades del polimorfismo, se transformará para poder responder a cada solicitud.



GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones comportamentales

Comando: El patrón comando permite aislar los objetos que realizan una petición de los objetos concretos encargados de recibir y realizar dicha acción, esto permite entre otras cosas, que las peticiones puedan ser enviadas a varios receptores y si se maneja el estado de las solicitudes, controla acciones de tipo Undo y Redo.

El patrón comando necesita la implementación de varios elementos:

- Comando: clase que sirve de puente entre el cliente y los receptores.
- Invoker: elemento usado por los clientes y que le solicita al comando llevar a cabo una acción.
- Cliente: invoca la ejecución de las acciones desde el Invoker.
- IComando: interfaz donde se especifican las operaciones a ejecutar.
- Receptor: clase que realiza la acción.
- Ejecutar: operación que necesita ser llevada a cabo.

GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones comportamentales

Comando: El patrón comando permite aislar los objetos que realizan una petición de los objetos concretos encargados de recibir y realizar dicha acción, esto permite entre otras cosas, que las peticiones puedan ser enviadas a varios receptores y si se maneja el estado de las solicitudes, controla acciones de tipo Undo y Redo.

Ejemplo: una persona (Cliente) quiere hacer uso del televisor y hace todas las solicitudes de servicios por medio del control remoto (*Invoker*). El control remoto se comunica con una interfaz encargada de responder solicitudes de cada comando, que el usuario puede hacer como, por ejemplo, prender televisor, apagar televisor, subir volumen, etc. Cada comando realiza una acción particular sobre el televisor (Receptor).



GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones comportamentales

Iterador: Este patrón de diseño está orientado al trabajo con colecciones y facilita el acceso a todos los elementos de la colección sin tener la necesidad de conocer su estructura.

1. En este patrón se reconocen los enumeradores y los iteradores. Los enumeradores se establecen para conocer el siguiente elemento de la estructura y los iteradores corresponden a un mecanismo para recorrer la estructura, de acuerdo con su secuencia, de inicio a fin.

2. El enumerado por lo tanto, se encarga de implementar un conjunto de métodos estándar para poder establecer la secuencia con la que se debe recorrer la estructura.

3. Los métodos más comunes: ***moveNext()*** que indica si existe o no un próximo elemento por recorrer. Método ***Current()*** que devuelve el valor actual de la colección según la posición actual en la secuencia. Método ***Reset()*** que permite iniciar nuevamente la secuencia a su punto de partida. El iterador necesita del enumerador para poder hacer el proceso de recorrido.

4. Este tipo de operaciones son tan comunes en los sistemas actuales, que los lenguajes de programación ya poseen una implementación propia del patrón iterador.

GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones comportamentales

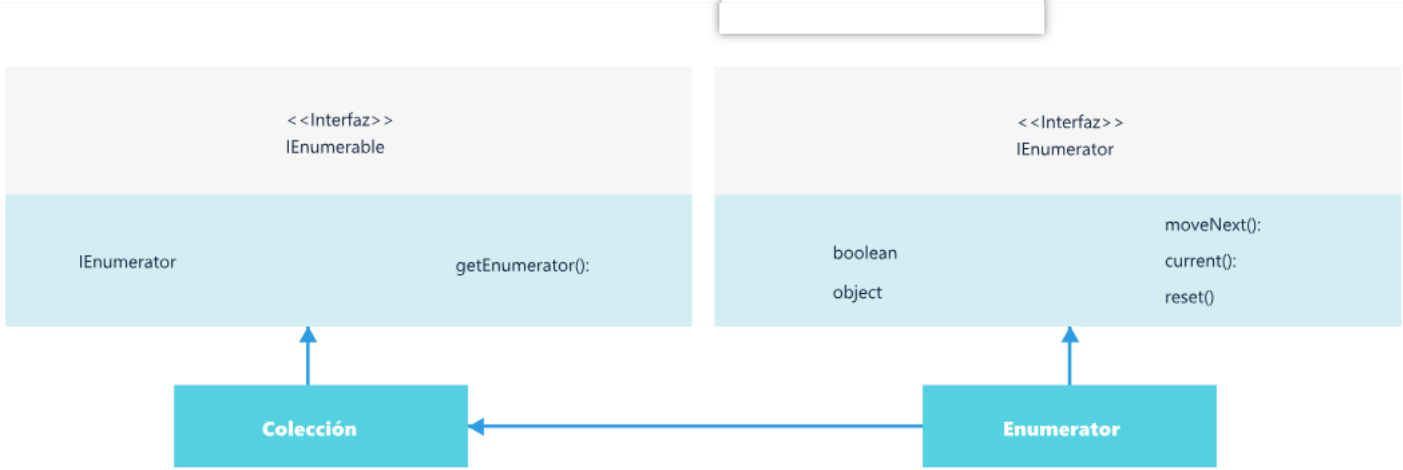
Iterador:

1. En este patrón se reconocen los enumeradores y los iteradores. Los enumeradores se establecen para conocer el siguiente elemento de la estructura y los iteradores corresponden a un mecanismo para recorrer la estructura, de acuerdo con su secuencia, de inicio a fin.

2. El enumerado por lo tanto, se encarga de implementar un conjunto de métodos estándar para poder establecer la secuencia con la que se debe recorrer la estructura.

3. Los métodos más comunes: ***moveNext()*** que indica si existe o no un próximo elemento por recorrer. Método ***Current()*** que devuelve el valor actual de la colección según la posición actual en la secuencia. Método ***Reset()*** que permite iniciar nuevamente la secuencia a su punto de partida. El iterador necesita del enumerador para poder hacer el proceso de recorrido.

4. Este tipo de operaciones son tan comunes en los sistemas actuales, que los lenguajes de programación ya poseen una implementación propia del patrón iterador.



GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones creacionales

Una de las labores comunes en el proceso de construcción de software es precisamente distribuir responsabilidades en un conjunto de módulos o clases, siguiendo los principios definidos en los paradigmas de programación, como por ejemplo, la programación orientada a objetos. Sin embargo, al momento de dar solución a un requerimiento particular se requerirá instanciar objetos de diferentes tipos, los cuales bajo sus responsabilidades implementan algún tipo de lógica.

En el siguiente recurso audiovisual puede profundizar los aspectos más importantes de los patrones creacionales: https://youtu.be/_KZkbL0MMbQ

GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones estructurales

Los patrones estructurales proveen una orientación relacionada a la forma de definir los componentes de los objetos. Dentro de los patrones estructurales se encuentran:

El patrón Fachada: utilizado cuando el sistema está compuesto por varios subsistemas.

El patrón *Delegate:* usado cuando se quiere reutilizar y extender funcionalidades de una clase, sin hacer uso de la herencia.

En este patrón se reconocen tres partes fundamentales. Landa, (2018):

Fachada: clase que provee las operaciones de alto nivel que serán usadas por el cliente.

Subsistemas: clases que proveen las funcionalidades que son expuestas por la fachada.

Cliente: hace uso de las operaciones de alto nivel por medio de la fachada.

GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones estructurales

Fachada: El patrón fachada se utiliza cuando el sistema está compuesto por varios subsistemas y se hace complejo gestionar los mensajes que debe realizar el cliente en cada uno de estos subsistemas. Este patrón permite generar al cliente una vista de alto nivel que simplifica el control y el envío de mensajes a los subsistemas, ocultando los detalles relacionados con la gestión de las clases e instancias.

Existen diferentes variaciones del patrón fachada:

1

Opaco

Una de las variaciones más utilizadas y en la cual los clientes no pueden acceder a los subsistemas, solo se puede hacer mediante el objeto fachada.

2

Transparente

El cliente tiene la posibilidad de acceder a los subsistemas por medio de la fachada, pero también puede hacerlo de forma directa.

3

Estática

En esta variación la fachada se implementa como una clase estática, por lo cual, no se hace necesaria la instancia de un objeto concreto de la fachada. Landa, (2018).

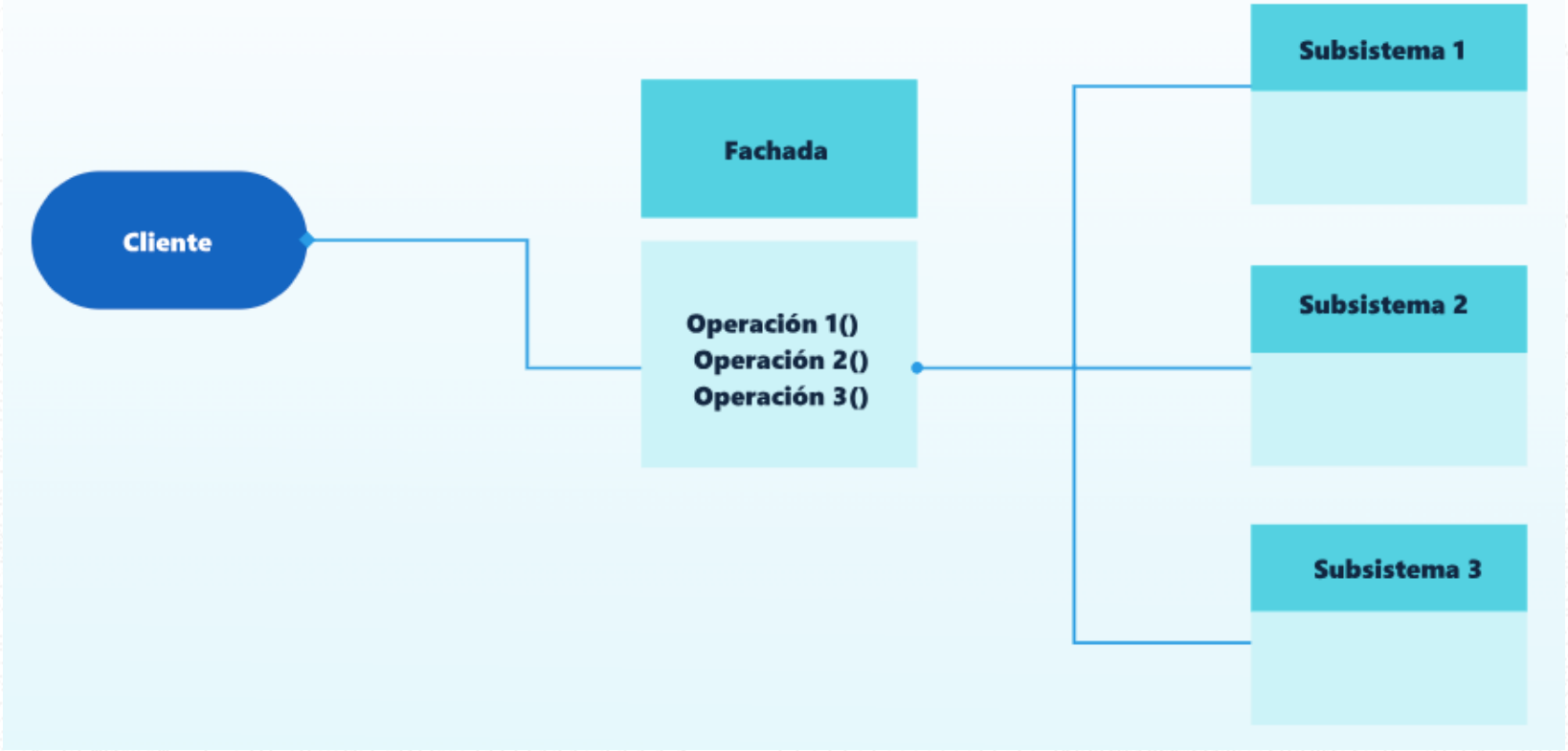
GA4-220501095-AA2-EV05

Conceptos sobre patrones de diseño – Patrones estructurales

Fachada: El patrón fachada se utiliza cuando el sistema está compuesto por varios subsistemas y se hace complejo gestionar los mensajes que debe realizar el cliente en cada uno de estos subsistemas. Este patrón permite generar al cliente una vista de alto nivel que simplifica el control y el envío de mensajes a los subsistemas, ocultando los detalles relacionados con la gestión de las clases e instancias.

Figura 1

Diagrama patrón fachada



GA4-220501095-AA2-EV05

Evidencia de desempeño: GA4-220501095-AA2-EV05 - Desarrollar la arquitectura de software de acuerdo al patrón de diseño seleccionado

- Incorporar patrones de diseño propendiendo en mejores prácticas para la codificación y mantenibilidad del software.
- **Elaborar la vista de componentes** para visualizar el software en fases avanzadas del ciclo de vida.
- **Elaborar la vista de despliegue** del software para determinar condiciones de la implantación de la solución informática.
- Elegir herramientas necesarias para optimizar los procesos.

Pista clave 1: Estudiar el componente formativo "*Diseño de patrones de software*".

Pista clave 2: **Video tutorial de apoyo**: https://youtu.be/sp8APOAyTfk

Pista clave 3: Implementación patrón fachada - archivos completos
https://gitlab.com/jonathanga/patron-fachada-en-java/-/tree/master

Estructura del documento:

Normas básicas de presentación de un documento escrito, debe contener como mínimo una portada, introducción, objetivos, desarrollo del trabajo y conclusiones.

G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682

@SENAComunica

www.sena.edu.co