**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | ADSO |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | **220501095. Diseñar la solución de software de acuerdo con procedimientos y requisitos técnicos.** | RESULTADOS DE APRENDIZAJE | **220501095-01. Elaborar los artefactos de diseño del software siguiendo las**  **prácticas de la metodología seleccionada.** |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 010 |
| NOMBRE DEL COMPONENTE FORMATIVO | **Diseño de patrones de *software*** |
| BREVE DESCRIPCIÓN | **Con el estudio de este componente, el aprendiz estará en capacidad de conceptualizar, interpretar y aplicar diseños de patrones de *software.* Así mismo, se afianzará en lo referente a patrones comportamentales, creacionales, estructurales, vistas estáticas, diagramas de despliegue y de componentes, todo ello en el marco general de análisis y desarrollo de *software.*** |
| PALABRAS CLAVE | Diseño, *hardware*, patrones y *software* |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | **Ciencias naturales, aplicadas y relacionadas** |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**

1. Conceptos sobre patrones de diseño

2. Patrones comportamentales

3. Patrones creacionales

4. Patrones estructurales

4.1. Fachada

4.2. *Delegate*

5. Vistas estáticas

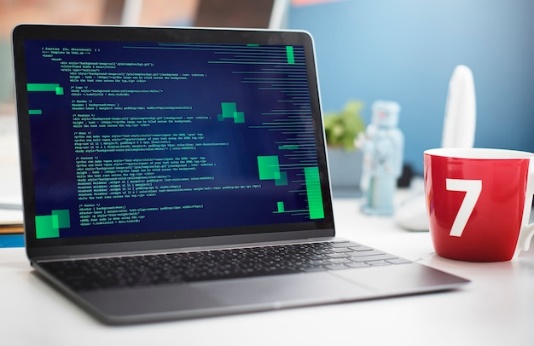
6. Diagrama de despliegue

7. Diagrama de componentes

1. **INTRODUCCIÓN**

**Los patrones de diseño surgen, inicialmente, en el área de la arquitectura y la ingeniería civil, refiriéndose a casos particulares a los que se enfrentaron en algún momento y de los cuales se documentó una solución particular. Posteriormente, este concepto se empezó a asociar al diseño orientado a objetos. En lugar de referirse a la forma de colocar paredes, puertas y ventanas, se hizo referencia a la forma en que se construyen clases, objetos e interfaces, y la forma en cómo estos deben interactuar.**

**Un patrón de diseño debe estar estructurado por cuatro componentes:**



**En este componente formativo, se abordarán algunos de los patrones de diseño más representativos en la industria del *software*, los cuales le permitirán implementar, a futuro, soluciones más robustas acordes con los altos estándares de calidad ampliamente conocidos en la industria. Este estudio profundizará en cómo estos patrones facilitan la reutilización de soluciones probadas y eficientes, mejorando así la mantenibilidad y la escalabilidad del *software.***

1. **DESARROLLO DE CONTENIDOS:**
2. **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:**

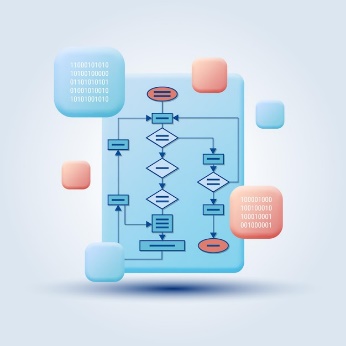
|  |
| --- |
| **Infografía estática**  **CF010\_1\_Conceptos sobre patrones de diseño** |

**Para comprender mejor la organización y aplicación de los patrones de diseño en el desarrollo de *software,* es importante reconocer su clasificación. Estos suelen dividirse en tres grandes grupos, según su finalidad:**

1. **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.**

* **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. Landa. (2018).**

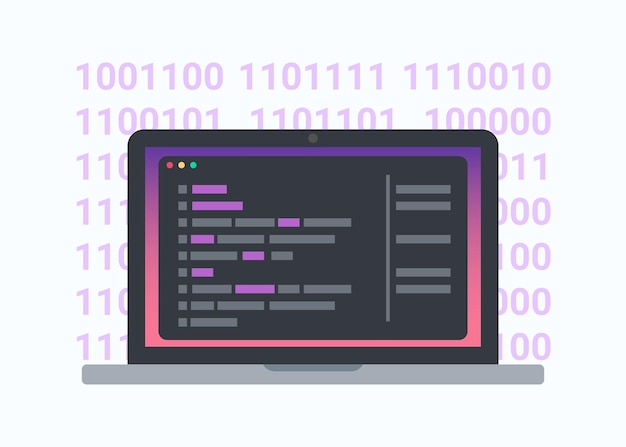
**Este patrón de diseño es útil cuando una misma funcionalidad puede ser provista usando diferentes mecanismos, algoritmos o estrategias, que serán seleccionadas dependiendo de las acciones realizadas por el cliente, en el momento que está ejecutando el programa.**

|  |
| --- |
| **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.**  **La imagen representa el ejemplo para el patrón comportamental Estrategia.** |

* **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:**

|  |
| --- |
| **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 el televisor, apagar el televisor, subir el volumen, entre otros. Cada comando realiza una acción particular sobre el televisor (Receptor).**  **La imagen describe el ejemplo del patrón comportamental Comando.** |

* **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.**



**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. El enumerado se encarga de implementar un conjunto de métodos estándar para poder establecer la secuencia con la que se debe recorrer la estructura.**

**Los métodos más comunes:**

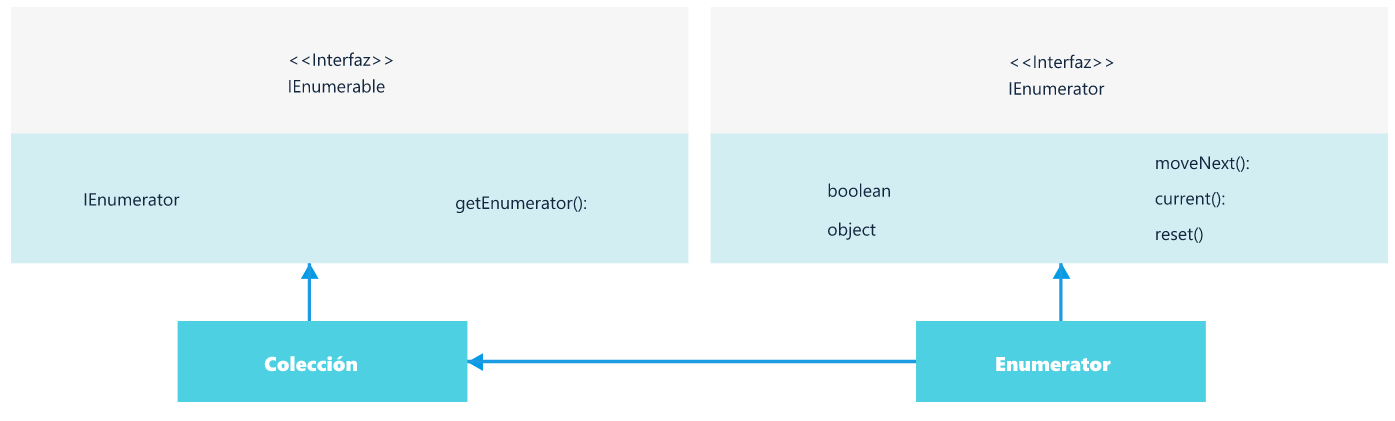


* + ***MoveNext(*) que indica si existe o no un próximo elemento por recorrer.**
  + **Método *Current()* que devuelve el valor actual de la colección según la posición actual en la secuencia.**
  + **Método *Reset()* que permite iniciar nuevamente la secuencia a su punto de partida.**

**El iterador necesita del enumerador para poder hacer el proceso de recorrido. Este tipo de operaciones son tan comunes en los sistemas actuales, que los lenguajes de programación ya poseen una implementación propia del patrón iterador.**

**Figura 1.**

**Ejemplo para el patrón comportamental iterador**

****

1. **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 recurso que le presentamos a continuación, profundice en los aspectos más importantes de los patrones creacionales:**

****

1. **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:**

**4.1 Fachada**

**El patrón fachado se emplea cuando un sistema está conformado por múltiples subsistemas, lo que complica la gestión de las interacciones que el cliente debe realizar con cada uno de ellos. 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:**

|  |
| --- |
| **Tarjetas**  **CF010\_4.1 Fachada (1)** |

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

**Se representa de la siguiente manera:**

**Figura 2.**

**Diagrama patrón fachada**

**La imagen representa el patrón fachada.
1. Cliente.
2. Operación 10, operación 20, operación 30.
3. subsistema 1, subsistema 2, subsistema 3.
**

**A continuación, se muestra una implementación en código Java de cada uno de los elementos requeridos para la implementación del patrón fachada en un ejemplo sencillo.**



**Ejemplo: el cliente quiere realizar una compra, pero este proceso involucra acciones por parte de tres diferentes subsistemas:**

* **Subsistema de compras: verifica la viabilidad de la tarjeta para el pago.**
* **Subsistema de inventario: verifica si hay productos en *stock***
* **Subsistema de envíos.**

**Para disminuir la complejidad, de la cual no debería ocuparse el cliente, se creará una fachada que encapsula las acciones de todos los subsistemas involucrados y proveer una interfaz simple con la que el cliente podrá interactuar más fácilmente. Como se presenta a continuación:**

|  |
| --- |
| **Acordeón**  **CF010\_4.1 Fachada (2)** |

***4.2 Delegate***

 **El patrón *delegate* se usa cuando se quiere reutilizar y extender funcionalidades de una clase sin hacer uso de la herencia. Este patrón permite de cierta forma implementar algo similar a la herencia múltiple que no es admitido por algunos lenguajes de programación, pero adicionalmente permite tener un control más detallado sobre este proceso, ya que se puede ocultar parte de los elementos heredados o, incluso, compartir elementos que no son posibles de heredar bajo el mecanismo de herencia tradicional**

**.**

|  |  |
| --- | --- |
| Lenguaje de programación para desarrolladores web. | **Este patrón busca evitar la concentración de todas las responsabilidades en una única instancia, optando en cambio por delegar las tareas en otras instancias especializadas en resolver dichas funciones.** |

**El siguiente ejemplo se muestra la implementación del patrón de diseño en cuestión, donde se aplica el patrón *Delegate* para integrar en una clase concreta funcionalidades definidas en otras clases a través del uso de interfaces.**

|  |
| --- |
| **Acordeón**  **CF010\_4.2 Delegate** |

**Cualquier nueva clase concreta que implemente las interfaces originales podrá ser utilizada por la clase Empleado para su reutilización, simplemente pasando la instancia correspondiente que implementa la nueva lógica al parámetro del constructor.**

1. **Vistas estáticas**

**La vista estática está encargada de modelar los conceptos significativos del dominio de la aplicación desde sus propiedades internas y las relaciones existentes. Se denomina vista estática porque no modela el comportamiento del sistema ni muestra las variaciones que se puedan presentar por efecto del tiempo.**

**Los elementos fundamentales de la vista estática son las clases que describen los conceptos del dominio del problema y las relaciones que pueden ser de tipo:**

**Entre los diagramas de Lenguaje Unificado de Modelado (UML) que se utilizan para representar la vista estática del sistema encontramos (ITCA, 2021):**



* **Diagrama de clases.**
* **Diagrama de objetos.**
* **Diagramas de componentes.**

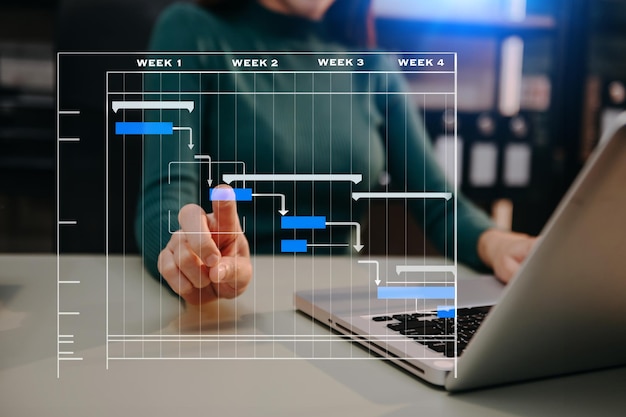
1. **Diagrama de despliegue**

**Los diagramas de despliegue, que forman parte de los tipos de diagramas propuestos por UML, tienen como objetivo representar la arquitectura física del sistema, tanto en *hardware* como en *software,* así como las conexiones entre ellos. Estos diagramas son de gran utilidad para el proceso de despliegue del sistema.**

**Los diagramas de despliegue utilizan un conjunto de elementos gráficos que tienen una representación y significado estandarizado. A continuación, se detalla cada uno de ellos:**

|  |
| --- |
| **Slides**  **CF010\_** **6\_Diagrama de despliegue** |

1. **Diagrama de componentes**

**El diagrama de componentes, uno de los diagramas sugeridos por UML, muestra una perspectiva estática del sistema de información y pertenece a los diagramas estructurales. Representa cómo se organiza y se relaciona el *software* en términos de componentes lógicos o físicos, como bibliotecas, módulos, paquetes y capas. Ofrece una visión general de los componentes del sistema, resaltando la organización y dependencia entre ellos, lo cual es fundamental para comprender cómo se integran y cooperan para realizar las funcionalidades del sistema.**

**Le invitamos a revisar los aspectos más característicos e importantes de los diagramas de componentes, explorando el recurso que se muestra a continuación:**

****

1. **SÍNTESIS**

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.

A diagram of a company

Description automatically generated

1. **ACTIVIDADES DIDÁCTICAS (Se debe incorporar mínimo 1, máximo 2)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Características representativas en patrones de diseño. |
| Objetivo de la actividad | Afianzar las características más importantes de algunos patrones de diseño de *software.* |
| Tipo de actividad sugerida | Arrastrar y solta |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF010\_Actividad didactica |

1. **MATERIAL COMPLEMENTARIO:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| **Conceptos sobre patrones de diseño** | **Leiva, A. (2020). Patrones de diseño software: Repaso completo en 10 minutos.** | **Video** | <https://www.youtube.com/watch?v=6BHOeDL8vls&feature=youtu.be> |
| **Diagrama de despliegue** | **Nicosiored. (2018). Diagrama de Despliegue - 22 - Tutorial UML en español** | **Video** | <https://www.youtube.com/watch?v=NSB0ATJUavA&feature=youtu.be> |
| **Diagrama de componentes** | **Nicosiored. (2018). Diagrama de Componentes I - 20- Tutorial UML en español [Video]. YouTube.** | **Video** | <https://www.youtube.com/watch?v=oOycG_n1ARs&feature=youtu.be> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| **Patrones GOF:** | **los patrones GOF, se trata de los patrones de diseño y se denominan de esa manera por el libro *Desing Patterns elements of reusable software*, de cuatro autores que descubrieron la manera fundamental de adentrarse en el mundo de la programación. GOF es la sigla de “*The Gang of four*”, que significa La pandilla de los cuatro.** |
| ***Software:*** | **se trata del compendio de rutinas o de programas que favorecen a los dispositivos el cumplimiento y ejecución de determinadas tareas, funciones o aplicaciones.** |
| **Lenguaje Unificado de Modelado (UML):** | **relaciona un conjunto de diagramas estandarizados para la representación de sistemas de información desde diferentes tipos de vista.** |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

**Creately (2021, Enero 15). La Guía Fácil de los Diagramas de Despliegue UML. [Web log post]. Blogspot.** [**https://creately.com/blog/es/diagramas/tutorial-de-diagrama-de-despliegue/**](https://creately.com/blog/es/diagramas/tutorial-de-diagrama-de-despliegue/)

**DiagramasUML (2013, Diciembre 8). ¿Qué es UML? ¿Qué diagramas componen UML? [Web log post]. Blogspot.** [**https://diagramasuml.com/componentes/**](https://diagramasuml.com/componentes/)

**EcuRed (2021). Patrones Gof. ECURED.** [**https://www.ecured.cu/Patrones\_Gof**](https://www.ecured.cu/Patrones_Gof)

**Escuela especializada en ingeniería (2021). Diagramas UML estáticos. ITCA-FEPADE.** [**https://virtual.itca.edu.sv/Mediadores/ads/222\_diagramas\_uml\_estticos.html**](https://virtual.itca.edu.sv/Mediadores/ads/222_diagramas_uml_estticos.html)

**Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Booch, G. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.**

**Landa, N. (2018). Patrones de Diseño de Software [Video]. YouTube.** [**https://www.youtube.com/playlist?list=PLM-p96nOrGcbqbL\_A29b0z3KUXdq2\_fpn**](https://www.youtube.com/playlist?list=PLM-p96nOrGcbqbL_A29b0z3KUXdq2_fpn)

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  (Para el SENA indicar Regional y Centro de Formación) | Fecha |
| Autor (es) | **Jonathan Guerrero Astaiza** | **Experto Temático** | **Regional Cauca - Centro de teleinformática y producción industrial.** | **Septiembre 2021** |
| Paola Alexandra Moya | Evaluadora instruccional | **Regional Antioquia** - Centro de Servicios de Salud | Marzo 2024 |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | **Regional Antioquia** - Centro de Servicios de Salud | Marzo 2024 |

1. **CONTROL DE CAMBIOS**

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |