PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Análisis y desarrollo de software
- Código del Programa de Formación: 228118 v1
- Nombre del Proyecto: Construcción de software integrando tecnologías a servicios
- Fase del Proyecto: Ejecución
- Actividad de Proyecto: Determinar la estructura lógica y tecnológica del sistema
- Competencia: Análisis de la especificación de requisitos del software.
- Resultados de Aprendizaje Alcanzar: Modelar las funciones del software de acuerdo con el informe de requisitos.
- Duración de la Guía: 26 horas

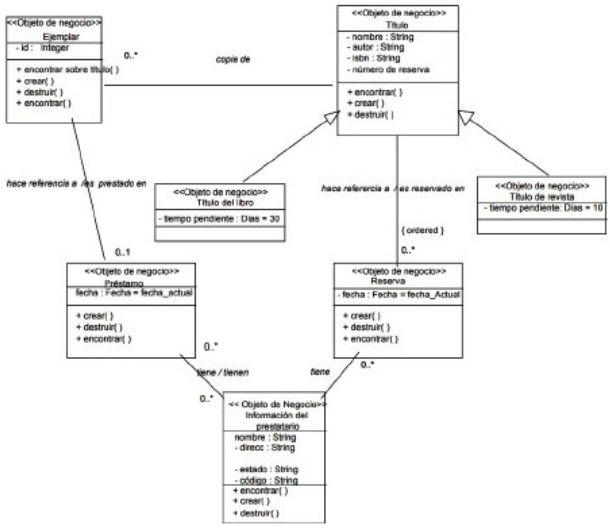
2. PRESENTACIÓN

¿Qué es y para qué sirve uml? El término “lenguaje” ha generado bastante confusión respecto a lo que es UML. En realidad, el término lenguaje quizás no es el más apropiado, ya que no es un lenguaje propiamente dicho, sino una serie de normas y estándares gráficos respecto a cómo se deben representar los esquemas relativos al software. Mucha gente piensa por confusión que UML es un lenguaje de programación y esta idea es errónea: UML no es un lenguaje de programación. Como decimos, UML son una serie de normas y estándares que dicen cómo se debe representar algo.

UML es una herramienta propia de personas que tienen conocimientos relativamente avanzados de programación y es frecuentemente usada por analistas funcionales (aquellos que definen qué debe hacer un programa sin entrar a escribir el código) y analistas-programadores (aquellos que, dado un problema, lo estudian y escriben el código informático para resolverlo en un lenguaje como Java, C#, Python o cualquier otro).

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

Diagrama de clases: Si los objetos tienen un comportamiento común o la misma estructura, pueden clasificarse (asignarse a una clase). La clase es, por tanto, un elemento simplificador (abstracción) para la representación visual. Las clases y los objetos están conectados entre sí mediante interfaces. El diagrama de clase muestra todos estos componentes y sus interrelaciones. Una clase representa el diagrama con un rectángulo. Contiene el nombre de la clase en negrita, como se muestra a continuación.



Te invitamos ahora a leer el documento "UML_Diagrama_de_Clases", que se encuentra ubicado en la carpeta: Material de Apoyo aprendices\Diagrama_de_Clases.pdf.

En donde, a partir de la lectura vas a diseñar un cuestionario de 10 preguntas cortas, cinco (5) de las respuestas de tipo única, y otras cinco (5) de falso o verdadero, luego comparte el cuestionario en clase para poder socializar entre todos.

Ambiente requerido: Ambiente de formación con equipos de cómputo y acceso a internet.

Materiales: Libreta de apuntes de cada aprendiz.

Realice en la herramienta Case UML (DIA, Visual Paradigm, Lucidchart, entre otras) de preferencia, el diagrama de clases, conforme al siguiente enunciado:

Caso de Estudio: Cadena de Venta de Electrodomésticos.

La cadena de venta de electrodomésticos Gabartino lanzará una nueva línea de créditos personales y sus directivos quieren minimizar el tiempo necesario para el otorgamiento de los mismos. Los potenciales compradores se acercan al mostrador de atención al cliente, donde pregunta que artículos son aptos para el otorgamiento de créditos. Si le gusta alguno de los artículos que se ofrecen, se acerca a un vendedor y le solicita que se lo muestre. Una vez que ha visto el producto en funcionamiento, el cliente puede decidir que el artículo no le resulta satisfactorio, en cuyo caso puede acercarse al mostrador de atención al cliente y ver si hay otro producto de su interés. También puede decidir irse del local sin comprar nada. En caso de que el artículo sea del agrado del cliente, el vendedor le pide sus datos y emite una orden de solicitud de crédito, la cual es entregada al departamento de venta. Los empleados de este departamento se encargan de estudiar la condición crediticia y fiscal del cliente a través del sistema SisCre. Simultáneamente se analiza su situación penal a través de un sistema de consulta provisto por la Interpol. Si alguno de los dos sistemas arroja elementos en contra del potencial cliente entonces el departamento de venta emite una orden de crédito rechazada. En caso contrario se emite una orden de crédito aprobada.

Ambiente Requerido: Ambiente de formación con equipos de cómputo y acceso a internet.

Materiales: Libreta de apuntes de cada aprendiz.

4. ACTIVIDADES DE EVALUACIÓN

Tome como referencia la técnica e instrumentos de evaluación citados en la guía de Desarrollo Curricular

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento: Identifica y conceptualiza cada uno de los componentes que corresponden a diagrama de caso de uso y diagrama de	Elabora propuestas de trabajo, de acuerdo con la interpretación de las necesidades tecnológicas, expuestas en el diseño de casos de uso y diagrama de clases.	Socialización, prueba de conocimientos / cuestionario diagrama de casos de uso y diagrama de clases.

clases. **Evidencias de Desempeño:** Analiza cada uno de los componentes que corresponden al diagrama de caso de uso y clases de su proyecto. **Evidencias de Producto:** Analiza de diagramas de casos de uso y clases y además realiza la documentación de los Diagramas de casos de uso correspondientes al proyecto.	Utiliza herramientas case para elaborar diagramas de casos de uso, que representen el estado actual de los componentes del sistema, apoyado en el análisis del informe de requerimientos. Elabora los diagramas UML, de acuerdo con las características de cada uno de ellos, basado en los requerimientos del cliente, utilizando herramientas case.	Sustentación Contextualización Documento de actividad de aprendizaje

5. GLOSARIO DE TÉRMINOS

Atributo: se refieren a las características que definen un servicio o producto. Los atributos se definen como tangibles (físicos) o intangibles (no físicos): Los tangibles son características como el tamaño, el color, el olor, empaque de producto, el diseño del producto, el peso, etc.

Artefacto: Objeto formado por un conjunto de piezas y fabricado para un fin determinado, en especial el que no constituye una máquina, aparato o dispositivo definidos.

Clase: es una construcción que permite crear tipos personalizados propios mediante la agrupación de variables de otros tipos, métodos y eventos.

Desplegue: refiere a desdoblar o extender lo que está plegado; a ejercitar o poner en práctica una actividad.

Extendido: son métodos estáticos, pero se les llama como si fueran métodos de instancia.

Flujo: pasos o procesos a seguir para alcanzar la solución de un problema.

Herencia: es un mecanismo de abstracción creado para poder facilitar y mejorar el diseño de las clases de un programa.

Include: Cualquier variable disponible en esa línea del archivo que hace el llamado, estará disponible en el archivo llamado, desde ese punto en adelante.

Método: es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase.

Modelo: Cosa que sirve como pauta para ser imitada, reproducida o copiada.

Operación: es un conjunto de normas y procesos que establece el programa.

Público: la clase puede ser utilizada por objetos que estén fuera del paquete actual.

Privado: especifica campos y métodos de una clase que no son accesible fuera de la unidad donde se declara la clase.

Protegido: que una clase y sus subclases tienen acceso a la variable, pero no a ninguna otra clase, necesitan usar un getter / setter para hacer cualquier cosa con la variable.

Relación: vínculo entre dos o más entidades describe alguna interacción entre las mismas.

Recurso: Asegurar una utilización eficiente y efectiva; Ganar confianza en que el cronograma es realístico.

Reservado: son aquellas palabras que tienen un significado predefinido en el lenguaje, por lo que no pueden.

Sistema: Conjunto ordenado de normas y procedimientos que regulan el funcionamiento de un grupo o colectividad.

Atributo: se refieren a las características que definen un servicio o producto. Los atributos se definen como tangibles (físicos) o intangibles (no físicos): Los tangibles son características como el tamaño, el color, el olor, empaque de producto, el diseño del producto, el peso, etc.

Artefacto: Objeto formado por un conjunto de piezas y fabricado para un fin determinado, en especial el que no constituye una máquina, aparato o dispositivo definidos.

Clase: es una construcción que permite crear tipos personalizados propios mediante la agrupación de variables de otros tipos, métodos y eventos.

Despliegue: refiere a desdoblar o extender lo que está plegado; a ejercitar o poner en práctica una actividad.

Extendido: son métodos estáticos, pero se les llama como si fueran métodos de instancia.

Flujo: pasos o procesos a seguir para alcanzar la solución de un problema.

Herencia: es un mecanismo de abstracción creado para poder facilitar y mejorar el diseño de las clases de un programa.

Include: Cualquier variable disponible en esa línea del archivo que hace el llamado, estará disponible en el archivo llamado, desde ese punto en adelante.

Método: es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase.

Modelo: Cosa que sirve como pauta para ser imitada, reproducida o copiada.

Operación: es un conjunto de normas y procesos que establece el programa.

Público: la clase puede ser utilizada por objetos que estén fuera del paquete actual.

Privado: especifica campos y métodos de una clase que no son accesible fuera de la unidad donde se declara la clase.

Protegido: que una clase y sus subclases tienen acceso a la variable, pero no a ninguna otra clase, necesitan usar un getter / setter para hacer cualquier cosa con la variable.

Relación: vínculo entre dos o más entidades describe alguna interacción entre las mismas.

Recurso: Asegurar una utilización eficiente y efectiva; Ganar confianza en que el cronograma es realístico.

Reservado: son aquellas palabras que tienen un significado predefinido en el lenguaje, por lo que no pueden.

Sistema: Conjunto ordenado de normas y procedimientos que regulan el funcionamiento de un grupo o colectividad.

Unificar: Hacer que varias cosas o personas distintas formen un todo o contribuyan a una causa común.

Hacer que varias cosas o personas distintas formen un todo o contribuyan a una causa común.

6. REFERENTES BILBIOGRÁFICOS

Gutiérrez, D. (2011). Casos de uso Diagramas de Casos de Uso. Gutiérrez, Damián, 1, 45.

ZAPATA, C., & TAMAYO, P. (2009). Generación del diagrama de casos de uso a partir del lenguaje natural o controlado: una revisión crítica. Dyna, 76(159), 193-203.

Garrido Tejero, A. (2021). Casos de uso y diagramas de casos de uso.

Peña, D. M., & Baquero, L. (2016). Extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso. Univ. las Ciencias Informáticas.

MUÑETÓN, A., ZAPATA, C. M., & ARANGO, F. (2007). Reglas para la generación automática de código definidas sobre metamodelos simplificados de los diagramas de clases, secuencias y máquina de estados de UML 2.0. Dyna, 74(153), 267-283.

Gutiérrez, D. (2011). UML Diagramas de Clases. Univ. los Andes Venez, 3, 1-19.

Zapata, C. M., & González, G. (2008). Especificación formal en OCL de reglas de consistencia entre los diagramas de clases y casos de uso de UML y el modelo de interfaces. Revista Ingenierías Universidad de Medellín, 7(12), 169-191.

7. CONTROL DEL DOCUMENTO

	Nombre	**Cargo**	**Dependencia**	**Fecha**
Autor (es)	Raúl Gómez	Instructor	centro de comercio y servicios	05/02/2024

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	**Cargo**	**Dependencia**	**Fecha**	**Razón del Cambio**
Autor (es)					

3. DIAGRAMAS DE CLASES 19

3.1. INTRODUCCIÓN 19
3.2. DIAGRAMAS DE CLASES 19
3.2.1. Perspectivas 20
3.2.2. Clases 20
3.2.2.1. Compartimento del nombre 21
3.2.2.2. Compartimento de la lista de atributos 21
3.2.2.2.1. Ejemplos de atributos 22
3.2.2.3. Compartimento de la lista de operaciones 22
3.2.2.3.1. Ejemplos de operaciones 23
3.2.3. Relaciones en un diagrama de clases 23
3.2.3.1. Asociación 24
3.2.3.1.1. Tipos de asociación 25
3.2.3.1.2. Diferencias entre los tipos de asociación 26
3.2.3.1.3. Ejemplos de asociaciones normales 26
3.2.3.1.4. Ejemplos de agregaciones 28
3.2.3.2. Dependencia 29
3.2.3.2.1. Ejemplos de dependencias 30
3.2.3.3. Generalización 31
3.2.3.3.1. Ejemplos de generalizaciones 31
3.2.4. Notas 32
3.2.5. Restricciones 33
3.2.6. Clases avanzadas (clasificadores) 33
3.2.6.1. Interfaces 34
3.2.6.2. Paquetes 35

3. DIAGRAMAS DE CLASES

3.1. INTRODUCCIÓN

La notación de UML está compuesta por dos subdivisiones importantes. Hay una notación para modelar los elementos estáticos tales como clases, atributos y relaciones. También hay otra notación para modelar los elementos dinámicos tales como objetos, mensajes y máquinas de estado finitas.

Los elementos estáticos se representan mediante diagramas de estructura estática, más conocidos por su nombre corto, diagramas de clases. Muestran el conjunto de clases que forman parte de la estructura estática de un sistema, junto con las relaciones existentes entre estas clases. Pero cuando se modela la estructura estática de un sistema, podemos usar los diagramas de clases de tres formas diferentes:

1. para modelar el vocabulario de un sistema

Modelar el vocabulario de un sistema implica tomar una decisión sobre qué abstracciones forman parte del sistema y qué otras caen fuera de sus límites. Aquí usamos los diagramas de clases para especificar dichas abstracciones y sus responsabilidades.

2. para modelar colaboraciones de forma sencilla

Una colaboración es un conjunto de clases, interfaces y otros elementos que trabajan juntos para proporcionar un comportamiento de cooperación mayor que la suma de todos los elementos. Por ejemplo, cuando queremos modelar las semánticas de una transacción en un sistema distribuido, no podemos fijarnos en una sola clase para entender lo que está sucediendo. Es mejor que estas semánticas sean llevadas a cabo por un conjunto de clases que trabajen juntas. Aquí usamos los diagramas de clases para visualizar y especificar este conjunto de clases y sus relaciones.

3. para modelar el esquema lógico de una base de datos

En muchos dominios podemos querer almacenar información continua (*persistent information*) en una base de datos relacional u orientada a objetos. Aquí usamos los diagramas de clase para modelar los esquemas de dichas bases de datos.

Nosotros vamos a estudiar los diagramas de clases que se utilizan para modelar el vocabulario de un sistema.

3.2. DIAGRAMAS DE CLASES

Los diagramas de clases se utilizan para modelar la visión estática de un sistema. Esta visión soporta los requisitos funcionales del sistema, en concreto, los servicios que el sistema debería proporcionar a sus usuarios finales. Normalmente contienen: clases, interfaces y relaciones entre ellas: de asociación, de dependencia y/o de generalización.

Los diagramas de clases también pueden contener a paquetes o subsistemas, que se usan para agrupar elementos del modelo en partes más grandes (por ejemplo, paquetes que a su vez contienen a varios diagramas de clases).

Al igual que otros diagramas, en los diagramas de clases pueden aparecer notas explicativas y restricciones.

3.2.1. Perspectivas

Según Fowler, hay tres perspectivas que podemos tener en cuenta a la hora de dibujar los diagramas de clases (o cualquier otro tipo de diagrama, pero es más evidente en los diagramas de clases):

- Conceptual: Estamos dibujando un diagrama que representa los conceptos del dominio del sistema. Estos conceptos se relacionarán de forma natural con las clases que los implementen, pero a menudo no hay una aplicación directa. Es decir, el modelo se dibuja sin tener en cuenta el software que lo implementa y generalmente es independiente del lenguaje de programación.

- Análisis: Desde el punto de vista del software, nos fijamos en las interfaces del software, no en su implementación. Es decir, miramos los tipos más que las clases.

  El desarrollo orientado a objetos pone mucho énfasis en la diferencia entre tipo y clase, pero luego no se aplica en la práctica. Es importante separar interfaz (tipo) de implementación (clase). Muchos lenguajes de programación no lo hacen y los métodos, influidos por ellos, tampoco. Esto está cambiando, pero no lo suficientemente rápido (Java y CORBA tendrán algo de influencia aquí). Los tipos representan una interfaz que puede tener muchas implementaciones diferentes debido, por ejemplo, a las características del entorno de instalación.

- Implementación: Estamos poniendo la implementación al descubierto, pues realmente tenemos clases. Es la perspectiva más utilizada, pero de todas formas la perspectiva del análisis se considera la mejor de las tres.

3.2.2. Clases

Las clases describen un conjunto de objetos con características y comportamiento idénticos, es decir, objetos que comparten los mismos atributos, operaciones y relaciones.

Las clases se representan gráficamente por medio de un rectángulo con tres divisiones internas. Los tres compartimentos alojan el nombre de la clase, sus atributos y sus operaciones, respectivamente. En muchos diagramas se omiten los dos compartimentos inferiores. Incluso cuando están presentes, no muestran todos los atributos y todas las operaciones. Por ejemplo, en la Figura 3.1 viene representada la clase Ventana de tres formas diferentes: sin detalles, detalles en el ámbito de análisis y detalles en el ámbito de implementación.

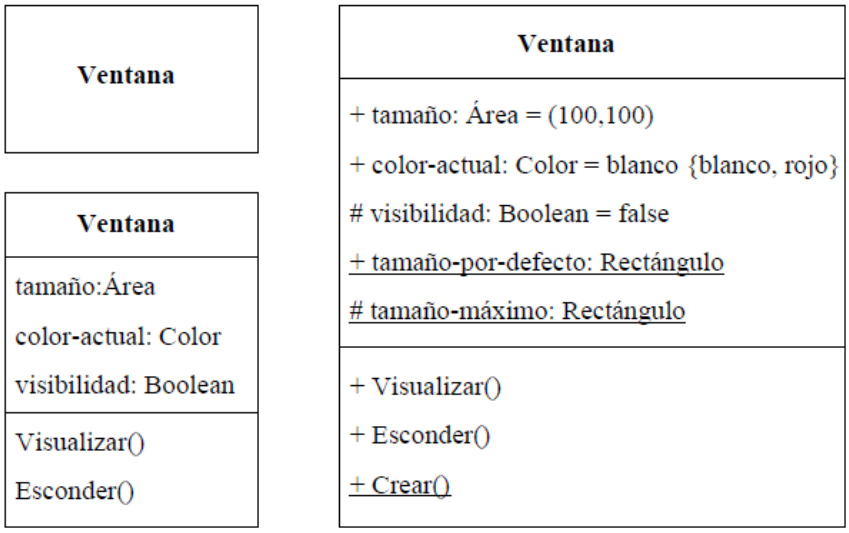


Figura 3.1

3.2.2.1. Compartimento del nombre

Cada clase debe tener un nombre que la distinga de las otras clases. Dicho nombre puede ser un nombre simple (nombre de la clase) o un nombre compuesto (nombre del paquete que contiene a la clase :: nombre de la clase).

En el ejemplo de la Figura 3.1 Ventana es un nombre simple de clase; pero si estuviese contenida en el paquete Gráficos, entonces el nombre compuesto sería Gráficos::Ventana. Otra forma de representar dicho nombre compuesto es escribir dentro de este compartimento primero el nombre del paquete (en este caso, «Gráficos») y debajo el nombre de la clase contenida en él.

3.2.2.2. Compartimento de la lista de atributos

Los atributos describen las características propias de los objetos de una clase. La sintaxis completa de un atributo es:

[visibilidad] nombre [multiplicidad] [: tipo] [= valor-inicial]
[{lista-propiedades}]

donde visibilidad puede ser:

- + (pública): que hace el atributo visible a todos los clientes de la clase;
- - (privada): que hace el atributo visible sólo para la clase;
- # (protegida): que hace el atributo visible a las subclases de la clase.

Si un atributo no tiene asignada ninguna visibilidad, quiere decir que la visibilidad no está definida (no hay visibilidad por defecto).

donde nombre es una cadena de texto que representa el nombre del atributo;

donde tipo es un tipo de atributo típico: string, boolean, integer, real, double, point, area y enumeration. Se llaman tipos primitivos. También pueden ser específicos de un cierto lenguaje de programación, aunque se puede usar cualquier tipo, incluso otras clases.

donde multiplicidad es un indicador de la multiplicidad del atributo, que va encerrado entre corchetes. La ausencia de multiplicidad significa que tiene exactamente valor 1, mientras que una multiplicidad de 0..1 proporciona la posibilidad de valores nulos (la ausencia de un valor).

donde valor-inicial es una expresión con el valor inicial que va a contener el atributo cuando se crea de nuevo el objeto;

donde lista-propiedades es una lista que contiene los valores permitidos en un atributo. Se utiliza para especificar tipos enumerados tales como color, estado, etc.

Ciertos atributos pueden ser visibles globalmente, en toda la amplitud léxica de la clase. Para ello se definen como atributos cuyo ámbito es la clase (*class-scope attributes*), llamados también variables de clase (*class variables*), y se representan como los objetos por un nombre subrayado. La notación se justifica por el hecho de que una variable de clase aparece como un objeto compartido por las instancias de una clase.

3.2.2.2.1. Ejemplos de atributos

En el ejemplo de la Figura 3.1 la clase Ventana tiene los siguientes atributos:

+ tamaño: Área = (100,100)
+ color-actual: Color = blanco {blanco, rojo}
# visibilidad: Boolean = false
+ tamaño-por-defecto: Rectángulo
+ tamaño-máximo: Rectángulo

Los atributos tamaño-por-defecto y tamaño-máximo tienen por tipo la clase Rectángulo, mientras que el resto corresponde a tipos primitivos: Área, Color y Boolean. Asimismo, estos dos atributos, que aparecen subrayados, son atributos cuyo ámbito es la clase.

3.2.2.3. Compartimento de la lista de operaciones

Las operaciones describen el comportamiento de los objetos de una clase. La sintaxis completa de una operación es:

[visibilidad] nombre [(lista-parámetros)] [: tipo-devuelto]
[{lista-propiedades}]

donde visibilidad puede ser:

- + (pública)
- - (privada)
- # (protegida)

Si una operación no tiene asignada ninguna visibilidad, quiere decir que la visibilidad no está definida (no hay visibilidad por defecto).

donde nombre es una cadena de texto que identifica la operación;

donde lista-parámetros es una lista de parámetros formales separados por una coma, cada uno especificado según la siguiente sintaxis:

nombre [: tipo] [= valor-por-defecto]

donde tipo-devuelto es una especificación del tipo que representa el valor devuelto por la operación. Depende del lenguaje de programación. Si la operación no devuelve ningún valor (por ejemplo, void en C++), se omite.

donde lista-propiedades es una lista que contiene los valores permitidos en una operación.

Una clase puede tener operaciones cuyo ámbito es la clase (*class-scope operations*). Aunque no exista un objeto de la clase, se puede llamar a una operación de este tipo, pero su acceso está restringido sólo a los atributos cuyo ámbito es la clase (*class-scope attributes*). Las operaciones cuyo ámbito es la clase se definen para llevar a cabo operaciones genéricas tales como crear objetos y encontrar objetos donde un objeto específico no está implicado. Se representan por un nombre junto con sus argumentos, todos ellos subrayados.

3.2.2.3.1. Ejemplos de operaciones

En el ejemplo de la Figura 3.1 la clase Ventana tiene las siguientes operaciones:

+ Visualizar()
+ Esconder()
+ Crear()

En estas operaciones no se ha especificado ninguno de los argumentos mencionados antes, excepto el de la visibilidad. La operación Crear es una operación cuyo ámbito es la clase, pues aparece subrayada.

3.2.3. Relaciones en un diagrama de clases

Los diagramas de clases están compuestos por clases y por relaciones entre ellas. Las relaciones que se pueden usar son:

- Relación de asociación

Una asociación es una conexión entre clases, una conexión semántica (enlace) entre los objetos de dichas clases. Un tipo especial de asociación es la relación de agregación.

- Relación de dependencia

Una dependencia es una relación entre elementos, uno independiente y otro dependiente. Un cambio en el elemento independiente afectará al elemento dependiente.

- Relación de generalización

Una generalización es una relación entre un elemento más general y otro más específico. El elemento más específico puede contener sólo información adicional. Una instancia (un objeto es una instancia de una clase) del elemento más específico se puede usar si el elemento más general lo permite.

3.2.3.1. Asociación

Una asociación es una relación estructural que especifica que los objetos de una clase están conectados con los objetos de otra. Cuando una asociación es una conexión semántica entre los objetos de dos clases, se llama asociación binaria. Aunque no es lo común, se pueden tener asociaciones que conecten más de dos clases; éstas se llaman asociaciones n-arias. También es posible que, dado un objeto de una clase, se pueda enlazar con otros objetos de la misma clase.

Normalmente una asociación es binaria y bidireccional (se puede navegar en ambas direcciones). Se dibuja como una línea sólida entre dos clases. También se puede representar una asociación binaria y unidireccional, añadiendo una flecha al final de la línea. Esta flecha indica que la asociación sólo se puede utilizar en la dirección de la flecha.

Cada asociación puede presentar algunos elementos adicionales que dan detalle a la relación, como son:

- Nombre

Una asociación puede tener un nombre, que se usa para describir la naturaleza de la relación. Así no hay ambigüedad sobre su significado. Para indicar la dirección en que se debe leer el nombre se emplea un triángulo.

- Rol

Cuando una clase participa en una asociación, juega un rol específico en dicha relación. Se puede designar de forma explícita mediante un nombre a los finales de la línea, el cual describe la semántica de la asociación en el sentido indicado.

- Multiplicidad

La multiplicidad describe la cardinalidad de la relación, es decir, cuántos objetos están conectados en una instancia de una asociación (enlace). Cuando se establece una multiplicidad al final de la línea de una asociación, indica que para cada objeto de la clase en el lado opuesto existen varios objetos en el otro extremo. El rango puede ser tres (3), cero-a-uno (0..1), cero-a-muchos (0..* ó *), uno-a-muchos (1..*), etc.

- Restricciones

Los elementos anteriores son suficientes para detallar una relación de asociación. Pero si queremos especificar un mayor significado, UML define cinco restricciones que se

pueden aplicar. Por ejemplo, si queremos que los objetos de una clase al final de una asociación (con una multiplicidad mayor que uno) estén en un orden explícito, debemos escribir {ordenado} al final de la misma y cerca de la clase.

En la Figura 3.2 se muestran los dos tipos de navegación en una asociación entre las clases Persona y Coche. En el primer caso un coche puede tener uno o más propietarios y una persona es propietaria de cero o más coches. Mientras que en el segundo caso la asociación navegable nos dice que una persona puede poseer varios coches o ninguno, pero no nos informa sobre cuántas personas son propietarias de un coche.

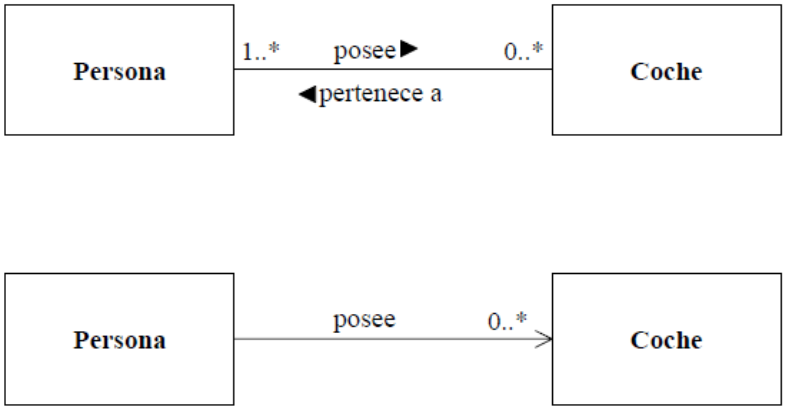


Figura 3.2

3.2.3.1.1. Tipos de asociación

Los tipos de asociaciones presentes en un diagrama de clases son:

1. Asociación normal

Una asociación normal entre dos clases representa una relación estructural entre sus objetos, lo que significa que ambas clases están conceptualmente al mismo nivel, ninguna es más importante que la otra. Se trata de una relación no muy fuerte.

2. Agregación

Una agregación sirve para modelar una relación "todo-parte", lo que significa que un objeto del todo tiene objetos de la parte. Podemos distinguir:

- Agregación normal

  Una agregación normal se denota dibujando una línea con un rombo sin rellenar al final de la misma del lado del todo (del lado de la clase que contiene a la otra clase). También puede tener un nombre (con dirección) y ser navegable, así como añadir roles en el lado de la parte (lado de la clase contenida).

Una agregación normal es un caso especial de la multiplicidad de una asociación, pues la cardinalidad del lado del todo sólo puede ser uno.

- Agregación compartida

Una agregación compartida es aquella en la que las partes pueden ser partes en varios todos. Se representa de la misma forma que una agregación normal. Para que una agregación se considere compartida es necesario que la multiplicidad en el lado del todo sea superior a uno.
Una agregación compartida es un caso especial de una agregación normal.

- Agregación de composición

Una agregación de composición se denota dibujando una línea con un rombo relleno al final de la misma del lado del todo (del lado de la clase que contiene a la otra clase). La multiplicidad en el lado del todo puede ser uno (1) ó cero-a-uno (0..1), pero la multiplicidad en el lado de la parte puede ser cualquier intervalo. Se trata de una relación de pertenencia muy fuerte.

3.2.3.1.2. Diferencias entre los tipos de asociación

La asociación expresa un acoplamiento débil; las clases asociadas siguen siendo relativamente independientes una de otra. Se trata de una relación no muy fuerte, es decir, no se exige dependencia existencial ni encapsulamiento.

La agregación es una forma particular de asociación que expresa un acoplamiento más fuerte entre clases; una de las clases cumple una función más importante que la otra en la relación. La agregación normal y compartida son relaciones no muy fuertes, con las mismas características que la asociación a la hora de implementarlas. Pero la agregación de composición es una asociación muy fuerte, que implica dependencia existencial (la clase dependiente desaparece al destruirse la que la contiene y, si es de cardinalidad uno, es creada al mismo tiempo) y pertenencia fuerte (la clase contenida es parte constitutiva y vital de la que la contiene). Una forma de implementar una agregación de composición es incluir físicamente las partes dentro de la clase del todo (encapsulamiento). En la forma de implementación es donde podemos hallar otra diferencia entre los tipos de agregaciones. Mientras que en la agregación de composición la clase del todo tiene el control del ciclo de vida de sus partes (al mismo tiempo que la clase es eliminada, se destruyen las partes), en las otras dos los tiempos de vida de la clase del todo y de sus partes no coinciden.

3.2.3.1.3. Ejemplos de asociaciones normales

En la Figura 3.3 podemos ver las clases y las relaciones de asociación normal, junto con sus elementos adicionales, correspondientes a una compañía de seguros. Observando el ejemplo, podemos darnos cuenta que:

- Una compañía de seguros tiene contratos de seguros, cada uno de los cuales se refieren a uno o más clientes.
- Un cliente tiene cero o más contratos de seguros, cada uno referido a una compañía de seguros.

- En consecuencia, un contrato de seguros se refiere a uno o varios clientes y a una compañía de seguros.
- El contrato de seguros está expresado en una (cero o una) póliza de seguros.
- La póliza de seguros se refiere a una compañía de seguros.

La multiplicidad especifica que el contrato de seguros está expresado en una (cero o una) póliza de seguros (contrato escrito) y que la póliza de seguros se refiere a una compañía de seguros. Si llamas por teléfono a la compañía y aseguras tu coche, hay un contrato de seguros, pero no una póliza. La póliza de seguros será enviada más tarde por correo.

Es importante modelar el mundo real. Si hubiésemos modelado la compañía de seguros basándonos sólo en la póliza de seguros, habrían surgido problemas. Por ejemplo, qué ocurriría si un cliente, que ha asegurado el coche por teléfono, tuviese un accidente con él un minuto más tarde (no hay póliza de seguros, pero sí un acuerdo oral con el agente).

Una clase puede jugar diferentes roles en diferentes asociaciones. En el diagrama de clases de la Figura 3.3, la clase Compañía de Seguros juega el papel de aseguradora y la clase Persona el papel de marido, mujer o asegurado.

Si nos fijamos en el ejemplo, todas las clases están conectadas mediante asociaciones binarias, excepto la clase Persona, donde un objeto de esta clase se puede enlazar con otros objetos de la misma clase. Un marido está casado con una mujer. Tanto el marido como la mujer son personas. Si una persona no está casada, entonces ni él ni ella juegan el papel de marido o mujer, lo que significa que la asociación (en este caso, casado con) no se aplica.

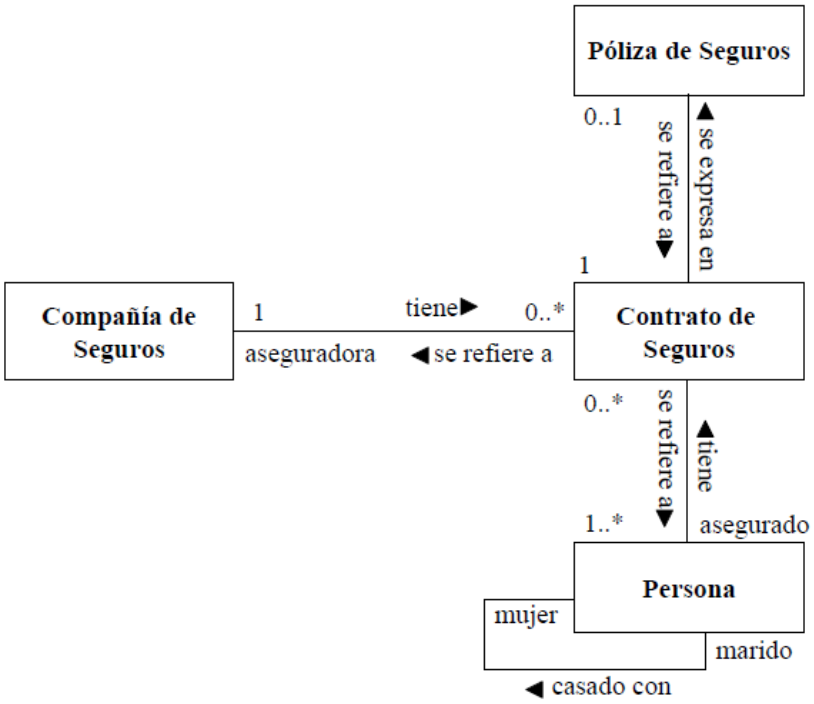


Figura 3.3

3.2.3.1.4. Ejemplos de agregaciones

En la Figura 3.4 podemos encontrar tres ejemplos con los tres tipos de agregaciones: normal, compartida y la de composición.

El primer ejemplo se trata de una agregación normal. La flota contiene muchos barcos de guerra. Algunos barcos pueden ser eliminados y todavía es una flota. Lo mismo ocurre si se añaden barcos, pues sigue siendo una flota. Esto es significativo para una agregación normal, pero no para una agregación de composición. Las partes (los barcos de guerra) componen el todo (flota). El rombo sin rellenar en el lado del todo, junto con la multiplicidad 1, nos indica que se trata de una agregación normal.

En el segundo ejemplo la agregación es compartida. Un remix (mezcla de sonidos) está compuesto de varias bandas sonoras; la misma banda sonora podría formar parte de varias mezclas musicales. Se diferencia de la agregación normal en que la multiplicidad en el lado del todo es superior a 1. Los objetos tanto de la clase Remix como de la clase Banda Sonora se encuentran ordenados, lo cual está indicado por la restricción {ordenado}.

El tercer ejemplo se trata de una agregación de composición. La ventana contiene muchos textos, cajas de listado de elementos, botones y menús. El tiempo de vida de las partes coincide con la clase del todo, en este caso, la clase Ventana. El rombo relleno en el lado del todo, junto con la multiplicidad 0..1 o 1, nos indica que se trata de una agregación de composición.

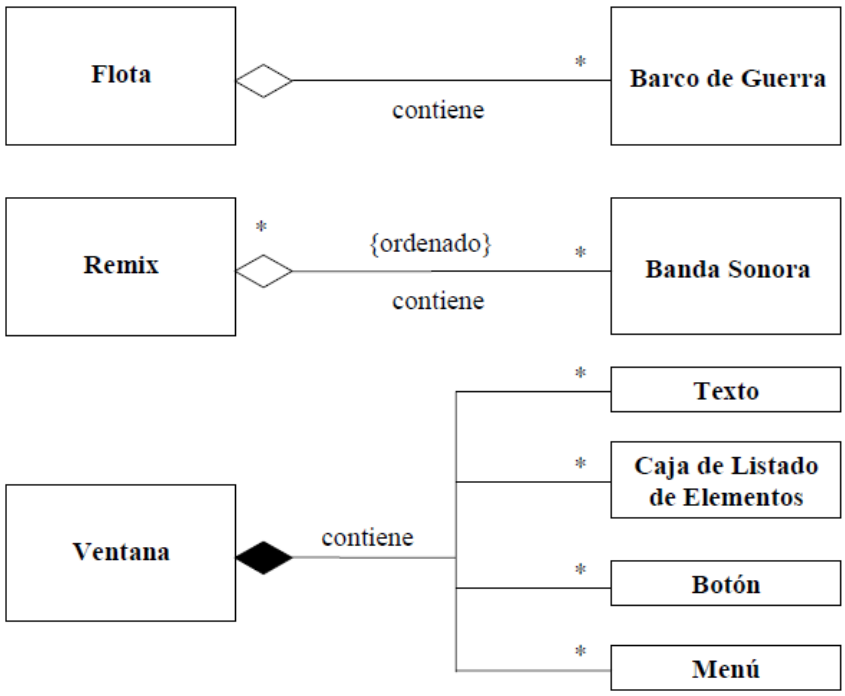


Figura 3.4

3.2.3.2. Dependencia

Una dependencia es una conexión semántica entre dos elementos de modelado, uno independiente y otro dependiente. Un cambio en el elemento independiente afectará al dependiente. Un elemento de modelado puede ser una clase, un paquete, un caso de uso, etc. Los casos más típicos de dependencia son: una clase tiene por parámetro en una operación un objeto de otra clase; una clase accede a un objeto global de otra clase; o una clase llama a una operación cuyo ámbito es la clase en otra clase.

La relación de dependencia se representa por una línea discontinua con una flecha en el lado del elemento independiente. Si se utiliza una etiqueta, entonces se trata de un estereotipo que identifica el tipo de dependencia.

En UML existen ocho estereotipos que se pueden aplicar a las relaciones de dependencia entre clases:

- bind

Se utiliza cuando se quiere modelar los detalles de las clases contenidas en plantillas. La relación entre una plantilla que contiene una clase y una instanciación de dicha clase se podría modelar como una dependencia *«bind»*. Bind incluye una lista de los argumentos actuales que se aplican a los argumentos formales de la plantilla.

- derive

Se utiliza cuando se quiere modelar la relación entre dos atributos o dos asociaciones, uno de los cuales es concreto y el otro es conceptual. Por ejemplo, la clase Persona podría tener el atributo fecha-de-cumpleaños (que es concreto), así como el atributo edad (que se puede derivar a partir del atributo anterior). Se podría mostrar esta relación mediante una dependencia *«derive»*, indicando que edad se deriva a partir de fecha-de-cumpleaños.

- friend

Se utiliza cuando se quiere modelar que una clase tenga acceso especial a la estructura interna de otra clase (incluso a aquellas partes con visibilidad privada). La dependencia *«friend»* es una reminiscencia de las clases amigas de C++.

- instanceOf
- instantiate

Estos dos estereotipos se utilizan para modelar de forma explícita las relaciones clase/objeto.
La dependencia *«instanceOf»* se utiliza cuando se quiere modelar la relación entre una clase y un objeto en el mismo diagrama, o entre una clase y su metaclase.
La dependencia *«instantiate»* se utiliza cuando se quiere especificar qué elemento crea objetos de otro.

- powertype

Se utiliza cuando se quiere modelar clases que encubran a otras clases, tal como sucede cuando se trata de modelar bases de datos.

- refine

Se utiliza cuando se quiere modelar clases que son esencialmente la misma, pero a diferentes niveles de abstracción. Por ejemplo, durante el análisis, tenemos una clase Cliente que, durante el diseño, vamos a refinar en una clase Cliente más detallada, es decir, más completa de cara a su implementación.

- use

Se utiliza cuando se quiere marcar de forma explícita una dependencia como una relación de uso, en contraste con los matices que proporcionan los otros estereotipos de dependencia.

3.2.3.2.1. Ejemplos de dependencias

La Figura 3.5 nos muestra un conjunto de cuatro clases junto con sus asociaciones y sus dependencias, correspondientes a una parte de un sistema que gestiona la asignación de profesores a los cursos en una universidad.

En el ejemplo hay una dependencia normal de la clase Planificación del Curso respecto a la clase Curso, pues Curso se utiliza como parámetro en las operaciones Añadir y Eliminar de Planificación del Curso.

Si proporcionamos la lista de parámetros en las operaciones, tal como se puede observar en el ejemplo, entonces no es necesario mostrar la dependencia, pues la utilización de la clase Curso ya figura de forma explícita en la clase Planificación del Curso. Sin embargo, a veces queremos que esta dependencia se dibuje, especialmente si hemos omitido la lista de parámetros en las operaciones.

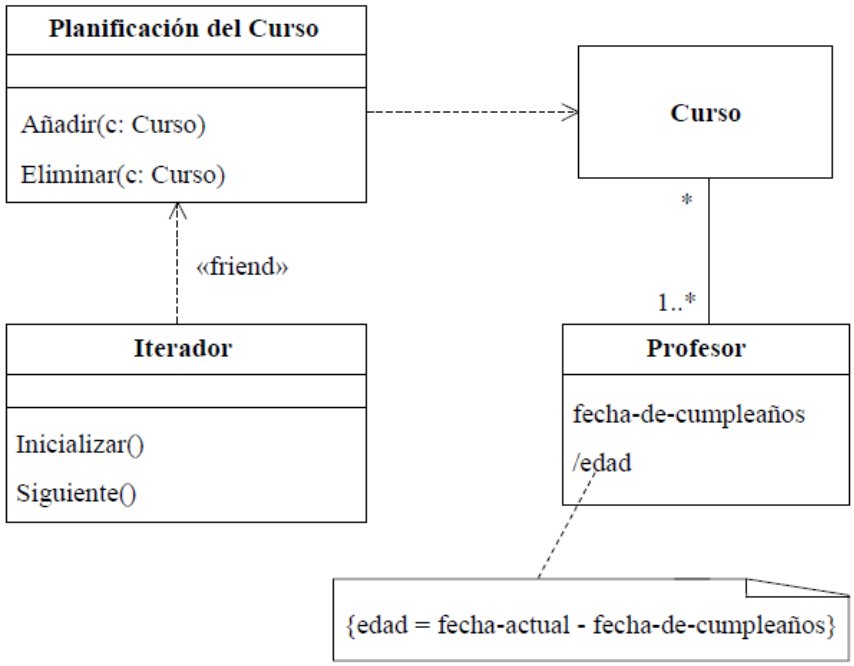


Figura 3.5

En el ejemplo aparecen otras dos dependencias marcadas con un estereotipo. Una de ellas es la dependencia *«derive»* entre los atributos edad y fecha-de-cumpleaños, la cual no se marca de forma explícita, sino como: /edad. La otra es la dependencia *«friend»* de la clase Iterador respecto a la clase Planificación del Curso. Esta dependencia muestra el uso de un iterador, especificado en la clase Iterador, sobre la programación del curso, especificada en la clase Planificación del Curso. Sin embargo, esta última clase no sabe nada acerca de la primera.

3.2.3.3. Generalización

Una generalización es una relación de clasificación entre un elemento más general y otro más específico. La generalización es utilizada por clases y casos de uso y por otros elementos de modelado como los paquetes.

La generalización normal es una relación de herencia entre clases, entre una general y otra específica. La clase específica, llamada subclase, hereda todo de la clase general, llamada superclase. Los atributos, las operaciones y todas las asociaciones son heredadas. Los atributos y las operaciones con visibilidad pública en la superclase serán también públicos en la subclase. Los atributos y las operaciones que tengan visibilidad privada también serán heredados, pero no serán accesibles dentro de la subclase. Para proteger los atributos y las operaciones de accesos desde fuera de la superclase y de la subclase, los podemos declarar con visibilidad protegida. No se puede acceder a ellos desde otras clases, pero están disponibles para la superclase y para cualquiera de sus subclases.

La relación de generalización se representa por una línea continua desde la clase más específica (subclase) hasta la clase más general (superclase), con un triángulo sin rellenar al final de la línea en la superclase.

Una clase abstracta es aquella que no tiene objetos. Sólo se utiliza para heredar a partir de ella, es decir, en una clase abstracta se describen los atributos y las operaciones comunes para otras clases. Se especifica de forma explícita poniendo {abstracta} dentro del compartimento del nombre de la clase y debajo de su nombre. Normalmente una clase abstracta tiene operaciones abstractas. Una operación abstracta es aquella que no tiene método de implementación en la superclase donde está definida. Se especifica de forma explícita escribiendo {abstracta} después del nombre de la operación en la superclase.

Lo contrario de una clase abstracta es una clase concreta, lo que significa que es posible crear objetos a partir de la clase y que tiene implementaciones para todas las operaciones.

A la hora de modelar aplicaciones sencillas es suficiente con la relación de generalización normal. Sin embargo, el modelado de ciertos dominios complejos (como puede ser el modelado de todas las aplicaciones de una empresa) requiere especificar un mayor significado, debido principalmente a la existencia de un gran número de subclases. Por ello UML define un estereotipo y cuatro restricciones que se pueden aplicar. El estereotipo es *«implementation»*. Las restricciones, que nos indican si se pueden añadir o no más hijos en la generalización, son: {completo} e {incompleto}. Las otras dos restricciones son: {disjoint} y {overlapping} y se aplican sólo en el contexto de herencia múltiple, es decir, cuando una clase tiene más de un padre.

3.2.3.3.1. Ejemplos de generalizaciones

La Figura 3.6 nos muestra un conjunto de cuatro clases junto con sus asociaciones y dependencias, referidas a tipos de vehículos.

En nuestro ejemplo la clase Vehículo muestra las cosas comunes entre coches y barcos, incluyendo la relación de asociación a la clase Persona. Se trata de una clase abstracta que no tiene objetos, mientras que las clases Coche y Barco son concretas. Estas subclases heredan el atributo color y la operación Conducir de la superclase Vehículo. Dicha operación está redefinida en las clases Coche y Barco, ya que no está implementada en la clase Vehículo al aparecer junto a la operación la palabra {abstracta} y además su implementación es diferente dependiendo del tipo de objeto: coche o barco.

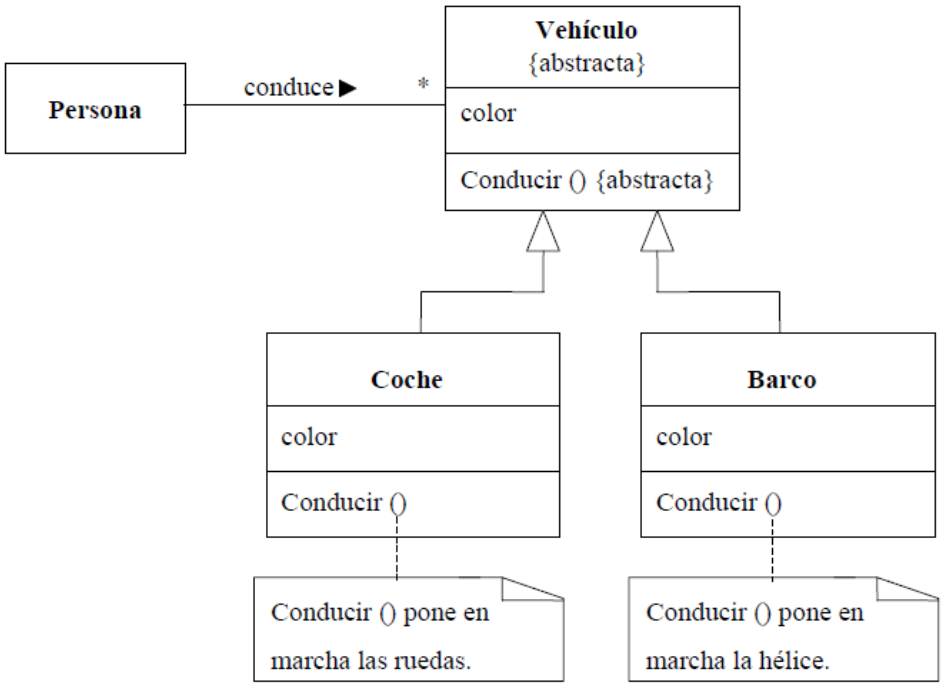


Figura 3.6

La Figura 3.6 nos muestra que la clase Persona tiene una asociación, llamada conduce, hacia la clase Vehículo. La clase Vehículo es abstracta; lo que significa que los objetos actuales que la persona conduce se obtienen a partir de las subclases concretas Coche y Barco. Cuando la persona ejecuta la operación Conducir, el resultado depende de si el objeto usado en dicha situación es un coche o un barco. Si es un objeto de la clase Coche, se pondrán en marcha las ruedas (utilizando la implementación especificada en la clase Coche). Si es un objeto de la clase Barco, se pondrá en marcha la hélice (utilizando la implementación especificada en la clase Barco). Esta técnica, donde un objeto de una subclase actúa según un objeto de una superclase y una o más operaciones en la superclase son redefinidas en la subclase, se llama polimorfismo.

3.2.4. Notas

Una nota es un comentario dentro de un diagrama cualquiera. Puede estar relacionado con uno o más elementos en el diagrama mediante líneas punteadas.

Una nota se representa gráficamente mediante un rectángulo con su borde superior derecho doblado. Sirve para representar aclaraciones al diagrama o restricciones sobre los elementos relacionados (cuando el texto se encuentre entre "{" y "}). Estas dos

representaciones se muestran en los ejemplos de la Figura 3.6 y la Figura 3.5, respectivamente.

3.2.5. Restricciones

Una restricción es una condición que limita el uso del elemento o las semánticas (significado) del elemento. Se declara como una cadena encerrada entre "{" y "}" y situada cerca del elemento asociado.

Las restricciones se pueden escribir como un texto en formato libre. Pero si queremos especificar las semánticas de forma más precisa, entonces debemos usar el lenguaje OCL (*Object Constraint Language*) de UML. De todas maneras, existe un número de restricciones ya definidas en UML que se pueden utilizar en cualquier diagrama.

3.2.6. Clases avanzadas (clasificadores)

Las clases constituyen el elemento de construcción más importante de cualquier sistema orientado a objetos. Sin embargo, las clases son uno de los tipos de un bloque de construcción más general en UML, los clasificadores.

Un clasificador es un mecanismo que describe las características de estructura y de comportamiento. UML proporciona un gran número de tipos de clasificadores, aparte de las clases, que nos pueden ayudar a modelar nuestro proyecto del mundo real.

- Interfaz: Un conjunto de operaciones que se utilizan para especificar un servicio de una clase o de un componente.
- Tipo de datos (*datatype*): Un tipo cuyos valores no tienen identidad, incluyendo tipos primitivos (tales como enteros y cadenas de caracteres), así como tipos enumerados (tales como booleanos).
- Señal (*signal*): La especificación de un estímulo asíncrono comunicado entre instancias.
- Componente: Una parte física de un sistema que proporciona la ejecución de un conjunto de interfaces.
- Nodo: Un elemento físico en tiempo de ejecución, que representa un recurso de ordenador, generalmente con memoria y capacidad de procesamiento.
- Caso de uso: Una descripción de un conjunto de acciones, incluyendo variantes, que un sistema lleva a cabo tras recibir un estímulo por parte de un actor en particular.
- Subsistema: Un grupo de elementos de los cuales algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos en el mismo.

Todos los clasificadores anteriores se representan gráficamente de distinta forma, tal como muestra la Figura 3.7.

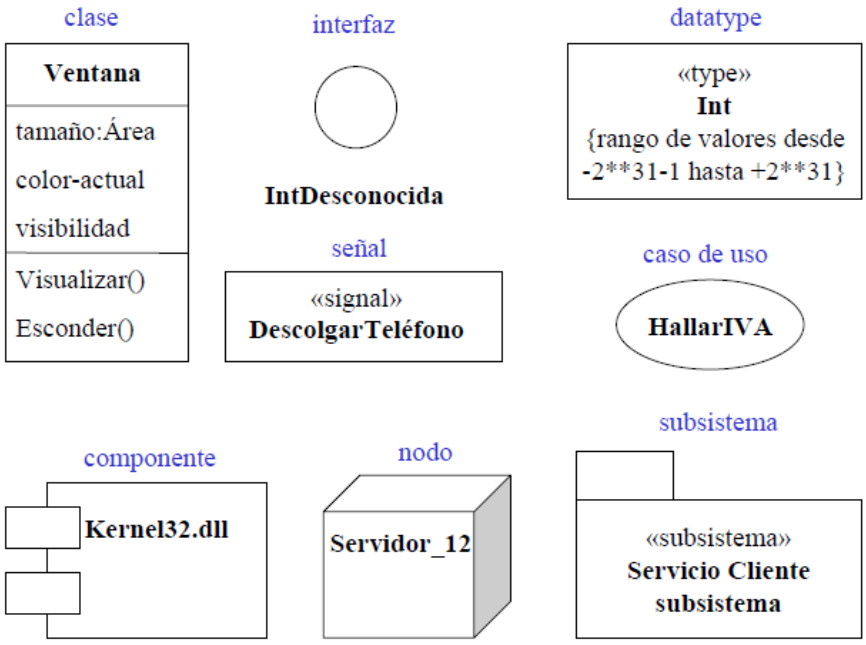


Figura 3.7

Un clasificador tiene características de estructura (en la forma de atributos), así como características de comportamiento (en la forma de operaciones). Las interfaces son la única excepción, pues no pueden tener atributos.

3.2.6.1. Interfaces

Una interfaz describe sólo operaciones abstractas, que especifican un comportamiento que un elemento (paquete, componente o clase) puede elegir soportar implementando la interfaz. A diferencia de las clases, las interfaces no especifican ninguna estructura (por ello no incluyen atributos) ni ninguna implementación (por ello no incluyen métodos, los cuales proporcionan la implementación de una operación). Estas operaciones abstractas pueden ser adornadas con propiedades de visibilidad y con restricciones. Al igual que las clases, las interfaces pueden participar en relaciones de asociación, dependencia y generalización.

Las interfaces se representan gráficamente por medio de un círculo pequeño con un nombre situado debajo de él. Se conectan a sus elementos mediante una línea continua, que representa una relación de asociación con multiplicidad 1-1.

Una clase que usa la interfaz implementada en una clase específica se conecta al círculo de dicha interfaz mediante una relación de dependencia (línea discontinua). La clase dependiente sólo depende de las operaciones de la interfaz, no depende de ninguna de las operaciones de la clase que la implementa (en cuyo caso, la relación de dependencia debería apuntar al símbolo de dicha clase). La clase dependiente podría llamar a las operaciones declaradas en la interfaz, las cuales no se muestran directamente en el diagrama. Para verlas es necesario que la interfaz sea especificada como una clase con el estereotipo *«interfaz»* dentro del rectángulo y por encima de su nombre.

En el primer ejemplo de la Figura 3.8 la clase Persona implementa la interfaz Almacenable.

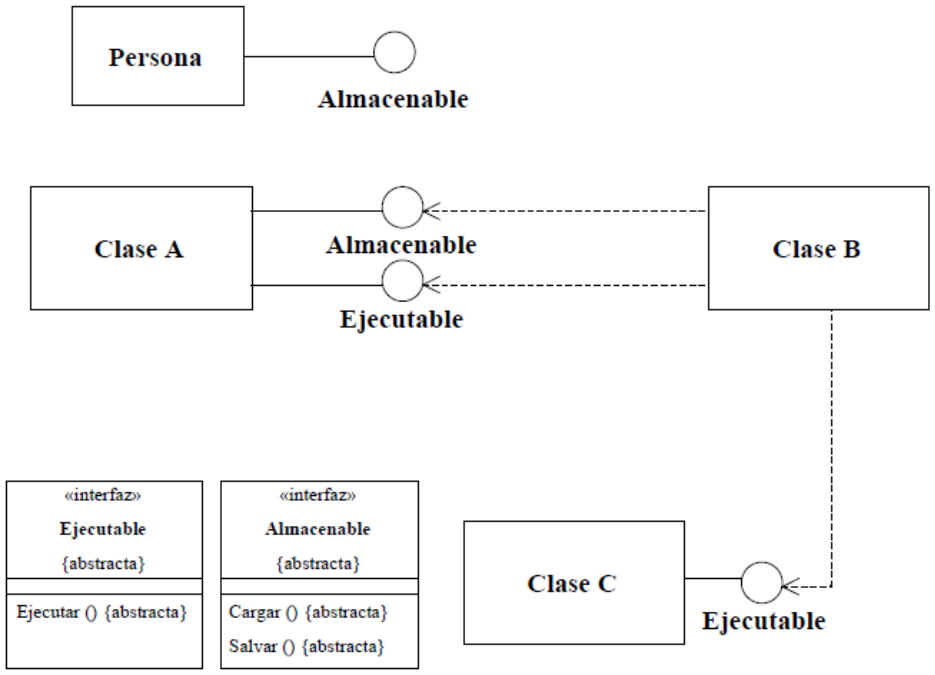


Figura 3.8

En el segundo ejemplo de la Figura 3.8 la clase A implementa las interfaces Ejecutable y Almacenable y la clase C la interfaz Ejecutable. La clase B usa las operaciones de las interfaces Ejecutable y Almacenable implementadas en A y las de Ejecutable implementada en C. Las interfaces son especificadas como clases con el estereotipo *«interfaz»* y contienen las operaciones abstractas que las clases que las implementan tienen que implementar.

3.2.6.2. Paquetes

Un paquete es un mecanismo para agrupar clases u otros elementos de otro tipo de diagramas en modelos más grandes, donde dichos elementos pueden ser linkados. Muchas metodologías de OO usan el término subsistema para describir un paquete.

Los paquetes o subsistemas se representan gráficamente por medio de un rectángulo grande junto con otro más pequeño situado en la esquina superior izquierda del rectángulo mayor. Si los contenidos de los paquetes no se muestran, entonces el nombre del paquete debe ir dentro del rectángulo grande; en caso contrario dentro del rectángulo pequeño.

Las relaciones permitidas entre paquetes son dependencia y generalización, tal como se muestra en la Figura 3.9.

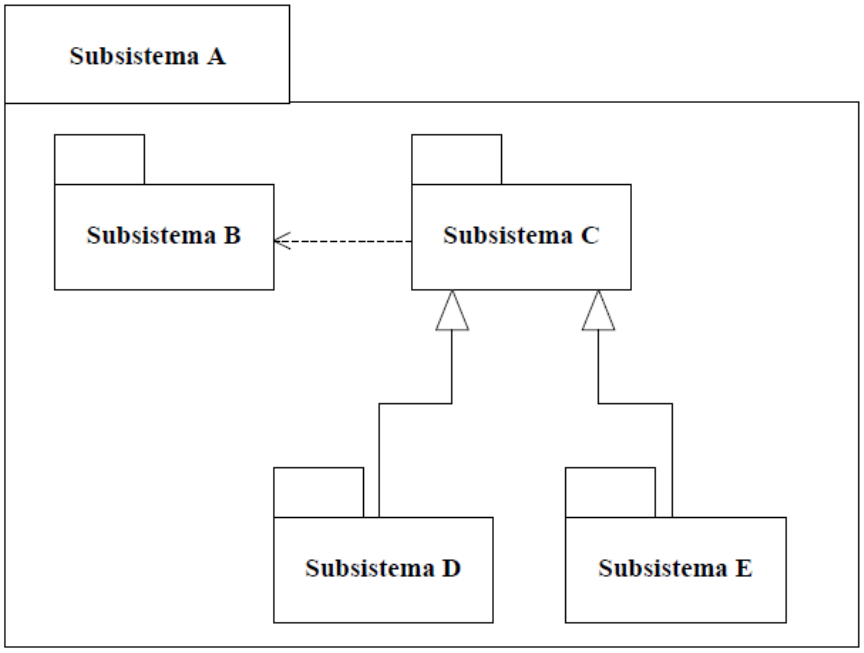

Subsistema A
Subsistema B
Subsistema C
Subsistema D
Subsistema E

Figura 3.9

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Análisis y Desarrollo de Software.
- Código del Programa de Formación: 228118
- Nombre del Proyecto Formativo (si aplica): Integración de tecnologías en el desarrollo de software.
- Fase del Proyecto (si aplica): Planeación.
- Actividad de Proyecto Formativo (si aplica): Estructurar procesos lógicos para la construcción de algoritmos.
- Competencia: Aplicación de conocimientos de las ciencias naturales de acuerdo con situaciones del contexto productivo y social.
- Resultados de Aprendizaje:
    - Solucionar problemas asociados con el sector productivo con base en los principios y leyes de la física.
    - Proponer acciones de mejora en los procesos productivos de acuerdo con los principios.
    - Verificar las transformaciones físicas de la materia utilizando herramientas
    - Identificar los principios y leyes de la física en la solución de problemas de acuerdo al contexto productivo.
- Duración de la Guía de Aprendizaje (horas): 40.

2. PRESENTACIÓN

La algoritmia es un área fundamental que estudia los algoritmos, secuencias de instrucciones definidas y finitas que permiten resolver problemas o llevar a cabo tareas específicas. Los algoritmos son una herramienta esencial en campos como la informática, las matemáticas, la ingeniería, y en realidad, en muchos aspectos de la vida cotidiana, desde la búsqueda de información en internet hasta la

realización de transacciones bancarias o la toma de decisiones logísticas. Podemos imaginar los algoritmos como una receta de cocina: siguen un conjunto de pasos bien definidos para lograr un objetivo, y la ciencia de la algoritmia se ocupa de diseñar, analizar y optimizar estos procesos.

Un algoritmo debe cumplir varias características: ser finito, es decir, terminar después de un número determinado de pasos; ser claro, con instrucciones precisas y sin ambigüedades; ser eficiente, utilizando la menor cantidad de recursos posibles; y tener entradas y salidas claramente definidas, proporcionando resultados efectivos. Estas características aseguran que los algoritmos sean herramientas poderosas para resolver problemas de manera eficaz.

La importancia de la algoritmia radica en que permite abordar problemas complejos de una manera estructurada. En un mundo donde se manejan grandes volúmenes de información y se requiere rapidez en la toma de decisiones, contar con algoritmos optimizados es clave. Por ejemplo, los motores de búsqueda como Google utilizan algoritmos para encontrar información relevante en cuestión de segundos. En logística, las empresas usan algoritmos para planificar rutas de entrega de manera óptima. En la seguridad informática, los algoritmos criptográficos protegen la información sensible, y en la inteligencia artificial, permiten que las máquinas aprendan y mejoren su desempeño con el tiempo. Por ello, la algoritmia no solo es una disciplina académica, sino una herramienta clave en la tecnología y el desarrollo actual.

Existen distintos tipos de algoritmos que se adaptan a diferentes problemas. Por ejemplo, los algoritmos de búsqueda permiten encontrar un elemento en una estructura de datos, mientras que los algoritmos de ordenación organizan conjuntos de datos de manera eficiente. Los algoritmos recursivos resuelven problemas dividiéndolos en subproblemas más pequeños, y los algoritmos de grafos son útiles para resolver problemas de redes, como encontrar el camino más corto en un mapa. También existen algoritmos de aproximación, que proporcionan soluciones cercanas al óptimo cuando una solución exacta es inviable debido a limitaciones de tiempo o recursos.

En el estudio de los algoritmos es fundamental el concepto de complejidad algorítmica, que mide los recursos que un algoritmo necesita para ejecutarse, principalmente tiempo y espacio. Se utiliza la notación de O grande para describir la complejidad en términos de cómo crece el tiempo de ejecución a medida que aumenta el tamaño del problema. Entender la complejidad de un algoritmo permite comparar distintas soluciones y seleccionar la más adecuada según el contexto. Esto es esencial cuando se trabaja con grandes volúmenes de datos, donde una diferencia en eficiencia puede representar una gran mejora en el desempeño.

El proceso de diseño de algoritmos comienza con un análisis detallado del problema a resolver. Una vez entendido el problema, se diseña una solución paso a paso, considerando las restricciones y objetivos. Después, se valida el algoritmo, asegurándose de que funcione correctamente en distintos escenarios. La implementación del algoritmo en un lenguaje de programación transforma la idea teórica en una herramienta funcional, y finalmente, se analiza la eficiencia del algoritmo, buscando formas de optimizar su rendimiento. Estas etapas permiten que los algoritmos no solo resuelvan problemas, sino que lo hagan de la manera más eficiente posible.

Para motivar el aprendizaje y fomentar el trabajo autónomo, es importante que los estudiantes vean la relación entre los algoritmos y su aplicación en la vida cotidiana. Por ejemplo, un algoritmo de



búsqueda binaria permite encontrar elementos en listas ordenadas de forma eficiente, lo que es la base de muchas funciones en bases de datos y motores de búsqueda. Algoritmos como Quicksort permiten ordenar grandes cantidades de datos rápidamente, y el algoritmo de Dijkstra, que encuentra el camino más corto en un grafo, es esencial en aplicaciones como los sistemas de navegación por GPS.

Relacionar el conocimiento previo con los nuevos aprendizajes es crucial para la apropiación de los conceptos. Los estudiantes pueden identificar cómo han utilizado la lógica y la toma de decisiones estructurada en otras áreas, como en la matemática básica o incluso en juegos de estrategia, y transferir esos conceptos al estudio de la algoritmia. Además, el aprendizaje colaborativo juega un rol importante en la resolución de problemas algorítmicos. Discutir soluciones en equipo, compartir ideas y enfrentarse a problemas complejos en grupo no solo refuerza el entendimiento individual, sino que también permite descubrir nuevas formas de abordar los desafíos.

El estudio de la algoritmia, además de ser una herramienta teórica, ofrece oportunidades para el desarrollo de proyectos prácticos donde los estudiantes pueden aplicar los algoritmos en situaciones del mundo real. Esto fomenta la transferencia de conocimientos y permite a los estudiantes conectar lo aprendido en el aula con problemas cotidianos, como la optimización de recursos en una empresa o el análisis de datos para la toma de decisiones. La práctica continua y la participación en desafíos de programación son excelentes formas de profundizar en el conocimiento de algoritmos, mejorando las habilidades de resolución de problemas y la capacidad de aplicar la teoría en contextos diversos.

La algoritmia es una herramienta poderosa para resolver problemas de manera eficiente, y su estudio permite desarrollar habilidades de pensamiento lógico, estructurado y crítico. Aprender algoritmia no solo es relevante para quienes desean dedicarse a la programación, sino para cualquier persona que busque mejorar su capacidad de análisis y resolución de problemas en su vida académica, profesional y personal. La clave está en relacionar los conceptos con la realidad, practicar de manera constante y colaborar en la resolución de problemas complejos para lograr una verdadera apropiación del conocimiento.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de reflexión inicial:

Descripción de la actividad:

Se inicia la formación explicando que los algoritmos no son solo un concepto abstracto o relacionado exclusivamente con la programación, sino que también están presentes en muchas actividades diarias. Después, se pregunta a los aprendices: ¿Qué es lo primero que les viene a la mente cuando escuchan la palabra “algoritmo”? Esta pregunta sirve como punto de partida para que cada uno comparta sus ideas iniciales, relacionándolas con sus experiencias o ideas previas sobre el tema.

Se plantea a los aprendices una situación cotidiana: “Piensa en alguna actividad rutinaria que realices todos los días, como vestirte, preparar un desayuno, o ir al trabajo o la escuela. Ahora, trata de describir los pasos que sigues para completar esa actividad.”

A partir de esta consigna, los estudiantes tienen unos minutos para pensar en su actividad y descomponerla en una serie de pasos. Se les pide que piensen en la claridad y el orden de esos pasos, identificando las posibles decisiones o bifurcaciones en el proceso. Por ejemplo, si alguien describe cómo prepara el desayuno, podría reflexionar sobre decisiones como: “Si hay leche en la nevera, preparo cereal; si no, hago una tostada.”

Los aprendices, en grupos pequeños, comparten la actividad que eligieron y los pasos que identificaron. Luego, cada grupo selecciona uno de los procesos descritos y lo expone al resto de la clase. Se guía la conversación, subrayando cómo cada uno de estos procesos puede interpretarse como un algoritmo, ya que consisten en una serie de instrucciones ordenadas para lograr un objetivo.

Después de escuchar las descripciones de los aprendices, se introduce formalmente el concepto de algoritmo, explicando cómo esas actividades cotidianas que acaban de describir son, en esencia, algoritmos básicos que seguimos para realizar tareas de manera eficiente. La reflexión inicial permite que los aprendices vean la algoritmia como algo más cercano y práctico, facilitando la transición hacia el estudio técnico del tema.

Se invita a los aprendices a reflexionar sobre la pregunta: “¿En qué otras áreas de tu vida crees que podrías aplicar el pensamiento algorítmico?” Esta reflexión final ayuda a consolidar la idea de que el pensamiento lógico y algorítmico es una habilidad útil en diferentes contextos y no solo en el ámbito de la programación o la informática.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.

Material de apoyo: Documento en PDF sobre algoritmia.

Duración de la actividad: 4 horas.

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Descripción de la actividad:

Se comienza explicando brevemente que un algoritmo no es solo un conjunto de pasos en la informática, sino una secuencia de instrucciones claras que permiten resolver problemas en cualquier contexto. Para establecer una conexión con la vida diaria y el conocimiento previo de los estudiantes, se plantea la pregunta: "¿Cómo crees que las tareas que realizas a diario pueden ser vistas como un proceso algorítmico?"

Se forma un mapa conceptual colectivo en el que los aprendices, proponen ejemplos de actividades cotidianas que involucran procesos algorítmicos. Estas actividades pueden ir desde organizar una lista de tareas hasta planificar un viaje o cocinar una receta. El mapa se va construyendo en el tablero o digitalmente, visualizando cómo cada una de estas actividades involucra una serie de pasos o decisiones estructuradas.

Se introduce la idea de que algunos de los pasos de estos procesos incluyen decisiones (por ejemplo, si una ruta está congestionada, elegir una alternativa) o repeticiones (como revolver una mezcla en la cocina hasta lograr una consistencia). A partir de aquí, se pregunta: "¿Conocen algún proceso que se resuelva a través de decisiones o repeticiones?" El objetivo es que los aprendices relacionen estos conceptos con actividades conocidas o experiencias previas, ya sea en situaciones cotidianas o en otras asignaturas, como matemáticas o ciencias.

Los aprendices, individualmente o en parejas, hacen una lista de ejemplos que conozcan donde se tomen decisiones (como bifurcaciones en un algoritmo) o se repitan pasos hasta alcanzar un resultado. Esta etapa permite descubrir qué tanto conocen sobre la lógica algorítmica sin haberla estudiado formalmente, ayudando a identificar áreas de fortaleza y aspectos que necesitarán más atención.

A continuación, se explica que las estructuras algorítmicas básicas como las decisiones (condicionales) y las repeticiones (bucles) son clave en la resolución de problemas computacionales. Se ejemplifica cómo, en el contexto de la programación, estas mismas ideas se transforman en instrucciones que las computadoras pueden seguir.

Los aprendices comparten sus ejemplos y experiencias, y se va guiando la discusión hacia la formalización del pensamiento algorítmico, resaltando cómo los conceptos que ya conocen pueden aplicarse en el ámbito de la programación. Esta parte del ejercicio ayuda a generar conexiones entre lo que los aprendices ya entienden de manera intuitiva y el nuevo conocimiento que aprenderán.

La actividad concluye con una síntesis donde se refuerza la idea de que muchos de los procesos cotidianos que los aprendices identificaron son en realidad algoritmos en acción. De esta manera, se contextualiza el tema y se identifican los conocimientos previos que servirán de base para aprender a estructurar algoritmos más formales y complejos.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.

Material de apoyo: Documento en PDF sobre algoritmia.

Duración de la actividad: 4 horas.

3.3 Actividades de apropiación:

Descripción de la actividad:

Se comienza revisando brevemente los conceptos clave que los aprendices han aprendido sobre algoritmos: la secuencia de pasos, las decisiones (condicionales) y las repeticiones (bucles). Explica que, en esta actividad, el objetivo es apropiarse de estos conceptos diseñando algoritmos para resolver problemas reales, utilizando lo que han aprendido en clases anteriores.

Se les presenta a los aprendices un problema realista que pueden resolver mediante la creación de un algoritmo. El problema debe estar relacionado con una situación cotidiana o con un contexto que les resulte familiar, para facilitar la conexión entre los conceptos teóricos y la vida práctica.

Los aprendices, organizados en grupos pequeños, deberán trabajar en la creación de un algoritmo que resuelva el problema presentado. Deberán pensar en cada uno de los pasos necesarios, tomando en cuenta las condiciones y las posibles decisiones o repeticiones que se requieren para llegar a la solución.

Se guía el proceso, respondiendo preguntas y ayudando a los aprendices a estructurar sus algoritmos de manera clara y lógica, pero dejando que sean ellos quienes propongan soluciones.

Después de haber diseñado el algoritmo, los grupos deben representarlo de manera visual, utilizando diagramas de flujo. Esto les ayudará a visualizar el proceso y a entender mejor la estructura de su algoritmo.

Cada grupo presenta su algoritmo al resto de la clase, explicando los pasos que han diseñado y cómo solucionan el problema. Los otros grupos pueden hacer preguntas o sugerencias para mejorar el diseño del algoritmo.

También se da retroalimentación constructiva, destacando qué aspectos del algoritmo son claros y bien definidos, y señalando posibles áreas de mejora en la eficiencia o en la estructura del proceso. El propósito es que los aprendices puedan refinar sus soluciones a partir de esta retroalimentación.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.

Material de apoyo: Documento en PDF sobre algoritmia.

Evidencias de aprendizaje: Taller fundamentos algoritmia.

Instrumentos de evaluación: Prueba de conocimientos.

Duración de la actividad: 4 horas.

3.4 Actividades de Transferencia el Conocimiento:

Descripción de la actividad:

Se comienza recordando que los algoritmos no solo se aplican en problemas técnicos o de programación, sino que también son útiles para resolver problemas en diferentes áreas como la organización de recursos, la toma de decisiones, la optimización de procesos, entre otros. Explica que, en esta actividad, el reto es transferir los conocimientos algorítmicos adquiridos en clase a un campo diferente al de la programación, usando los mismos principios lógicos.

Se proponen varios escenarios o áreas temáticas nuevas que requieren la aplicación de un algoritmo para resolver un problema. Estos escenarios pueden abarcar desde la gestión empresarial hasta la planificación de eventos o la optimización de tareas en un entorno real.

En grupos pequeños, los aprendices seleccionan uno de los escenarios propuestos y diseñan un algoritmo que resuelva el problema planteado, pero trasladando el conocimiento aprendido en las clases de algoritmia a este nuevo contexto.

Los aprendices deben trabajar de manera autónoma, aplicando sus conocimientos previos para formular la solución.

Una vez que cada grupo haya diseñado su algoritmo, deben discutirlo dentro del grupo, revisando si cumple con las condiciones del problema y si es eficiente. Se pide a los aprendices que revisen posibles mejoras y que optimicen su solución si encuentran formas de hacerlo. Esta etapa refuerza el aprendizaje colaborativo y fomenta el pensamiento crítico.

Cada grupo presenta su algoritmo al resto de la clase, explicando el problema que resolvieron y el proceso que siguieron para diseñar su solución. Otros grupos pueden hacer preguntas o sugerir mejoras.

Se proporciona retroalimentación, enfocándose en cómo los aprendices lograron transferir el conocimiento algorítmico a un nuevo contexto, evaluando la claridad de los pasos, la eficiencia del algoritmo y su adecuación al problema planteado.

La actividad concluye con una reflexión grupal, donde se plantea preguntas como: "¿Cómo este ejercicio les ayudó a entender que los principios algorítmicos pueden aplicarse más allá de la programación?" y "¿En qué otras áreas creen que podrían aplicar el conocimiento sobre algoritmos?".

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.

Material de apoyo: Documento en PDF sobre algoritmia.

Evidencias de aprendizaje: Taller estructuras secuenciales, taller estructuras selectivas y taller estructuras iterativas.

Instrumentos de evaluación: Prueba de conocimientos.

Duración de la actividad: 4 horas.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Planeación	Estructurar procesos lógicos para la construcción de algoritmos	Algoritmia	Taller estructuras secuenciales. Taller estructuras selectivas. Taller estructuras iterativas.	Da respuesta sobre conectores lógicos, entidades primitivas, algoritmos, fases de los algoritmos, estructuras y su aplicación	Prueba de conocimientos.

				en la solución de un problema. Representa procesos del sistema a partir de la construcción de algoritmos, como parte de la solución a situaciones planteadas. Realizar algoritmos utilizando variables, constantes, variables, bucles, contadores, acumuladores, selección, pseudocódigo.	

5. GLOSARIO DE TÉRMINOS

Lectura: Consiste en recibir desde un dispositivo de entrada (p.ej. el teclado) un valor.

Escritura: Consiste en mandar por un dispositivo de salida (p.ej. monitor o impresora) un resultado o mensaje.

Algoritmo: Secuencia de pasos lógicos para dar solución a un problema, con un inicio y fin definidos.

Variable: Espacio reservado de memoria cuyo valor puede cambiar durante la ejecución del programa.

Constante: Espacio reservado de memoria que permanece constante durante la ejecución del programa.

Diagrama de flujo de datos: Representación gráfica de un algoritmo.

Asignar: Establecer un valor a una variable.

Condicional: Estructura algorítmica de decisión que permite seguir un camino verdadero o falso en base a la o las condiciones.

Ciclo Anidado: Estructura algorítmica que consiste en ubicar un ciclo dentro de otro para realizar alguna tarea específica.

Index: Variable de tipo entera encargada de controlar (contar) el número de iteraciones o veces que se repite un ciclo.

Ciclo: Estructura algorítmica que permite repetir una o varias líneas de código según se cumpla una condición.

Valor de verdad: Unidades para validación de los procesos de la lógica, puede ser Verdadero o falso ó en su defecto Si o No ó 1 o 0.

6. REFERENTES BILBIOGRÁFICOS

Prieto, S. (sf). Principios de algoritmia.

Instituto tecnológico de Sonora (2016). Pseudocódigo y PSEINT.

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	Raúl Gómez	Instructor	Red de Informática y Sistemas	02/10/2024

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Análisis y Desarrollo de Software.
- Código del Programa de Formación: 228118
- Nombre del Proyecto Formativo (si aplica): Integración de tecnologías en el desarrollo de software.
- Fase del Proyecto (si aplica): Análisis.
- Actividad de Proyecto Formativo (si aplica): Especificar el modelo conceptual y estructurar la propuesta técnica del proyecto de software.
- Competencia: Razonar cuantitativamente frente a situaciones susceptibles de ser abordadas de manera matemática en contextos laborales, sociales y personales.
- Resultados de Aprendizaje:
    - Resolver problemas matemáticos a partir de situaciones generadas en el contexto social y productivo.
    - Identificar modelos matemáticos de acuerdo con los requerimientos del problema planteado en contextos sociales y productivo.
    - Proponer acciones de mejora frente a los resultados de los procedimientos matemáticos de acuerdo con el problema planteado.
    - Plantear problemas matemáticos a partir de situaciones generadas en el contexto social y productivo.
- Duración de la Guía de Aprendizaje (horas): 40.

2. PRESENTACIÓN

La lógica matemática es una herramienta fundamental para desarrollar el pensamiento crítico y la resolución de problemas. En este contexto, se presenta una introducción a la lógica matemática, cuyo estudio es clave en diversas áreas como la informática y las matemáticas. Además de proporcionar una

base sólida para comprender conceptos complejos, el aprendizaje de la lógica ayuda a mejorar habilidades de análisis y síntesis, lo que es esencial tanto en situaciones académicas como en la vida cotidiana.

La lógica matemática se enfoca en estudiar los principios que determinan la validez de los razonamientos. Entre los temas clave de esta disciplina se encuentran las proposiciones, los conectores lógicos, las tablas de verdad, las reglas de inferencia y los cuantificadores. Un ejemplo cotidiano que puede ayudar a motivar su estudio es mostrar cómo la lógica está presente en la toma de decisiones o en la estructura de los sistemas informáticos. Esto permite que los estudiantes relacionen los nuevos conceptos con experiencias concretas que ya conocen.

Un aspecto importante es la conexión entre la lógica matemática y conocimientos previos de los estudiantes. Por ejemplo, se puede relacionar con el álgebra booleana, utilizada en la construcción de circuitos lógicos, o con el lenguaje natural, al demostrar cómo los argumentos cotidianos pueden formalizarse utilizando reglas lógicas. Para reforzar esta conexión, es útil invitar a los estudiantes a reflexionar sobre situaciones en las que han aplicado razonamientos lógicos, como en juegos de estrategia o la resolución de problemas complejos.

Es fundamental motivar a los estudiantes a trabajar de forma autónoma. Proponer desafíos lógicos individuales les permite practicar y desarrollar el pensamiento crítico sin depender de ayuda inmediata. Resolver problemas, como tablas de verdad o validar argumentos mediante reglas de inferencia, no solo fortalece el conocimiento teórico, sino que también fomenta la autoconfianza y la capacidad de análisis.

El aprendizaje colaborativo es otro pilar en el estudio de la lógica matemática. Organizar actividades en grupo, donde los aprendices deban resolver problemas complejos o construir argumentos lógicos en equipo, les ayuda a aprender de sus compañeros, compartir ideas y enfrentarse a problemas desde distintas perspectivas. Este tipo de colaboración puede incluir debates sobre la validez de ciertos razonamientos, donde los estudiantes defienden o refutan argumentos utilizando las reglas de la lógica que han aprendido.

Al finalizar la sesión, es útil hacer un repaso de los conceptos clave, revisando las proposiciones, tablas de verdad y reglas de inferencia vistas anteriormente. A su vez, se puede enfatizar la importancia de continuar profundizando en el estudio de la lógica matemática, destacando cómo su dominio es esencial para estudios más avanzados en matemáticas, ciencias de la computación e incluso filosofía. Animar a los aprendices a continuar su aprendizaje de manera autónoma y colaborativa les permitirá seguir creciendo en su comprensión de esta disciplina.

Finalmente, para consolidar el conocimiento adquirido, se pueden proponer tareas individuales, como ejercicios de análisis de proposiciones y resolución de tablas de verdad, y recomendar recursos adicionales que permitan a los aprendices seguir explorando la lógica matemática por su cuenta. Estos recursos pueden incluir libros, tutoriales y videos que complementen el aprendizaje en el ambiente.

Este enfoque permite a los aprendicecs no solo comprender la teoría, sino también aplicarla en situaciones prácticas, desarrollando competencias tanto individuales como colaborativas en el proceso.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de reflexión inicial:

Descripción de la actividad:

Al inicio de la sesión, pide a los aprendices que piensen en un ejemplo reciente en el que hayan tenido que tomar una decisión o resolver un problema siguiendo un razonamiento lógico. Para guiar la reflexión, ofrece preguntas como:

¿Alguna vez has tenido que decidir entre varias opciones con base en condiciones específicas?

¿Cómo analizas la veracidad de una afirmación en una conversación o discusión?

¿Has jugado algún juego de estrategia que te obligue a pensar varios pasos adelante?

Después de unos minutos de reflexión individual, se invita a los aprendices a compartir sus ejemplos con la formación o en pequeños grupos. A medida que van contando sus experiencias, escribe en el tablero los diferentes tipos de razonamientos que se mencionan y cómo se relacionan con conceptos clave de la lógica matemática, como proposiciones, condiciones y deducciones.

Se concluye la actividad señalando que, aunque quizás no lo noten, la lógica está presente en muchas decisiones y problemas que enfrentamos en el día a día. Luego, conecta estas situaciones cotidianas con la importancia del estudio formal de la lógica matemática para afinar esas habilidades de razonamiento y aplicarlas de manera más eficiente en contextos académicos y profesionales.

Esta actividad no solo sirve para activar el conocimiento previo, sino que también motiva a los aprendices al mostrarles la relevancia práctica de lo que van a aprender.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.

Material de apoyo: Documento en PDF sobre lógica matemática.

Duración de la actividad: 4 horas.

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Descripción de la actividad:

Contextualizar la lógica matemática en distintos ámbitos de la vida diaria y académica, mientras los estudiantes identifican los conocimientos previos que poseen sobre el tema.

Se inicia la actividad planteando una pregunta amplia y reflexiva: "¿En qué áreas de la vida creen que utilizamos la lógica?" Los aprendices pueden pensar en cómo se aplica la lógica en la toma de decisiones, en juegos, en los razonamientos científicos o en el diseño de tecnología. También se puede proporcionar ejemplos de la lógica en distintos campos, como en los algoritmos que utilizan las aplicaciones móviles, los argumentos legales, o los procesos de diagnóstico en medicina.

A continuación, se organiza una lluvia de ideas en la que los aprendices compartan ejemplos concretos de situaciones donde creen que la lógica está presente. Se puede escribir estas ideas en el tablero o en una plataforma colaborativa, clasificando las respuestas en diferentes áreas: vida cotidiana, matemáticas, ciencias, tecnología, juegos, entre otros.

Una vez que se ha generado una lista de ejemplos, se realiza una pregunta: "¿Qué saben sobre cómo funciona la lógica? ¿Han escuchado sobre proposiciones, tablas de verdad o inferencias?" El objetivo es que los aprendices, individualmente o en grupos, hagan una breve reflexión sobre lo que ya conocen o han escuchado acerca de la lógica matemática. Se recopila estas respuestas para identificar el nivel de conocimiento del grupo.

Después de la discusión, se presentan algunos conceptos básicos de lógica matemática (como proposiciones, operadores lógicos y tablas de verdad) para que los aprendices puedan compararlos con lo que ellos mencionaron durante la lluvia de ideas. Aquí es útil señalar conexiones directas entre las ideas de los aprendices y los conceptos que se van a estudiar en profundidad durante el programa.

Finalmente, se presenta una breve explicación sobre la importancia de la lógica matemática en diferentes campos, vinculando los ejemplos aportados por los aprendices con el contenido formal de la asignatura. Se pueden hacer conexiones explícitas entre los ejemplos cotidianos que mencionaron y cómo estos pueden ser modelados y analizados de manera lógica y matemática.

Al concluir la actividad, se puede invitar a los aprendices a reflexionar sobre cómo la lógica puede ayudarlos a mejorar en distintas áreas de su vida académica y cotidiana. Se les plantea la pregunta: "¿Cómo creen que el estudio formal de la lógica matemática puede ayudarlos a resolver problemas más complejos?" Esto permite proyectar el aprendizaje hacia el futuro y motivar su estudio.

Al final de la actividad, los aprendices deben haber identificado ejemplos de la aplicación de la lógica en diferentes contextos, así como reconocer qué saben y qué les falta aprender. La reflexión final les permitirá anticipar cómo el estudio formal de la lógica matemática enriquecerá sus habilidades de razonamiento, tanto en su vida personal como en sus estudios académicos.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:
Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.
Material de apoyo: Documento en PDF sobre lógica matemática.
Duración de la actividad: 4 horas.

3.3 Actividades de apropiación:

Descripción de la actividad:

Se comienza la actividad presentando una situación o problema cotidiano que los aprendices deben resolver utilizando la lógica formal. Por ejemplo, se puede plantear un escenario en el que un grupo de amigos debe decidir entre ir al cine o hacer una excursión, dependiendo de ciertas condiciones climáticas y preferencias individuales. Las condiciones y restricciones estarán expresadas mediante proposiciones (p, q, r…) y conectores lógicos (“y”, “o”, “si… entonces”).

Los aprendices, de manera individual o en grupos, deben traducir el escenario cotidiano a proposiciones lógicas formales. Por ejemplo:

p: “Si hace buen tiempo, haremos la excursión.”
q: “Si hay lluvia, iremos al cine.”
r: “Si Pedro no puede asistir, no haremos la excursión.”

De este modo, los aprendices identifican y formulan claramente las condiciones en términos lógicos, utilizando proposiciones y conectores adecuados.

Una vez que han formulado las proposiciones, los aprendices construyen una tabla de verdad que refleje las distintas combinaciones de verdad y falsedad de las proposiciones, para analizar todos los posibles resultados. Este paso les permitirá visualizar las relaciones entre las diferentes proposiciones y sus conectores lógicos.

A partir de la tabla de verdad, los aprendices deben identificar las conclusiones posibles o inferencias que se pueden extraer del problema. Por ejemplo, si ciertas condiciones se cumplen, ¿es posible hacer la excursión o no? ¿Qué ocurre si Pedro no puede asistir, pero el tiempo es bueno? Mediante el uso de reglas de inferencia (Modus Ponens, Modus Tollens, etc.), los estudiantes validan si sus conclusiones son válidas o inválidas.

Tras completar las tablas y validar las inferencias, se organizan pequeños grupos de discusión en los que los aprendices comparan sus resultados. Se circula entre los grupos ofreciendo retroalimentación, planteando preguntas adicionales para guiar la reflexión: “¿Qué pasaría si una de las condiciones

cambiara? ¿Cómo afectaría eso al resultado?" Este diálogo fomenta la reflexión crítica y la revisión de su propio razonamiento lógico.

Para afianzar el proceso de apropiación, se propone un segundo problema de contexto distinto (por ejemplo, relacionado con la informática o la toma de decisiones en una empresa), en el que los aprendices deberán aplicar las mismas herramientas lógicas. Esto les permite ver que los conceptos de proposiciones, tablas de verdad y reglas de inferencia no son limitados a un solo contexto, sino que son herramientas universales.

Al final de la actividad, se pide a los aprendices que reflexionen sobre el proceso que siguieron y cómo la lógica les ayudó a analizar y resolver problemas de manera más estructurada. Preguntas como "¿Cómo te ayudó la lógica a tomar una decisión en este caso?" o "¿Cómo crees que podrías aplicar estos conceptos en otros campos de estudio?" facilitan la internalización de los conceptos y promueven una reflexión metacognitiva sobre su propio aprendizaje.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:
Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.
Material de apoyo: Documento en PDF sobre lógica matemática.
Evidencias de aprendizaje: Taller lógica matemática.
Instrumentos de evaluación: Prueba de conocimientos.
Duración de la actividad: 4 horas.

3.4 Actividades de Transferencia el Conocimiento:

Descripción de la actividad:

Se presenta un problema que implica diseñar un algoritmo simple para la gestión de acceso en un sistema informático. El algoritmo debe decidir si un usuario puede acceder a ciertos recursos basándose en varias condiciones lógicas (por ejemplo, permisos de usuario, horario de acceso, y ubicación). Cada condición deberá ser expresada mediante proposiciones lógicas.

Se describe el problema detalladamente. Por ejemplo, las condiciones para acceder a un sistema podrían ser las siguientes:

p: “El usuario tiene permisos de administrador.”
q: “El usuario accede dentro del horario permitido.”
r: “El usuario se encuentra en una red segura.”

Los aprendices deben traducir estas condiciones al lenguaje lógico-matemático utilizando conectores como “y” ($\wedge$), “o” ($\vee$) y “si… entonces” ($\rightarrow$). Deberán formular la lógica de acceso de modo que el sistema permita el acceso solo si se cumplen las condiciones específicas.

Con las proposiciones ya establecidas, los aprendices deberán crear una estructura lógica que pueda ser implementada en un pseudocódigo o lenguaje de programación (por ejemplo, usando sentencias condicionales como if, else y operadores lógicos AND, OR, NOT).

Una vez diseñado el algoritmo, los aprendices deben probarlo o discutir cómo funcionaría con distintos escenarios de entrada. Utilizan tablas de verdad para validar todas las combinaciones posibles de valores (verdadero/falso) para las proposiciones y verificar que el algoritmo responde correctamente a cada combinación.

Por ejemplo:

Si un usuario tiene permisos de administrador y está en horario permitido (p = verdadero, q = verdadero), el sistema debería conceder acceso.

Si el usuario no tiene permisos de administrador (p = falso), el acceso debe ser denegado, sin importar las otras condiciones.

A continuación, se les plantea a los aprendices un problema en un contexto diferente pero con una estructura lógica similar. Por ejemplo, pueden crear un sistema de recomendación para una tienda en línea, donde las condiciones serían:

p: “El cliente tiene historial de compra de productos similares.”
q: “El producto está en oferta.”
r: “El producto está en stock.”

El nuevo desafío consiste en diseñar un algoritmo que recomiende productos a los clientes según estas condiciones, aplicando nuevamente las mismas reglas de lógica matemática pero en un entorno distinto.

Después de implementar el segundo algoritmo, los aprendices comparan cómo las reglas lógicas que utilizaron en ambos contextos son similares y cómo el mismo principio lógico puede ser adaptado a diversas áreas. Esta discusión ayuda a que los aprendices interioricen la flexibilidad y poder de las herramientas lógicas.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:
Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Tablero, marcadores, borrador y equipo de cómputo.
Material de apoyo: Documento en PDF sobre lógica matemática.
Evidencias de aprendizaje: Taller tablas de verdad.
Instrumentos de evaluación: Prueba de conocimientos.
Duración de la actividad: 4 horas.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Análisis	Especificar el modelo conceptual y estructurar la propuesta técnica del proyecto de software	Lógica de programación	Taller lógica matemática. Taller tablas de verdad.	Da respuesta sobre tablas de verdad y conectores lógicos. Elabora Tablas de verdad de acuerdo con los problemas planteados.	Prueba de conocimientos.

5. GLOSARIO DE TÉRMINOS

Valor de verdad: Unidades para validación de los procesos de la lógica, puede ser Verdadero o Falso ó en su defecto Si o No ó 1 o 0.

Algoritmo: Secuencia de pasos lógicos para dar solución a un problema, con un inicio y fin definidos.

6. REFERENTES BILBIOGRÁFICOS

Prieto, S. (sf). Principios de algoritmia.

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	Raúl Gómez	Instructor	Red de Informática y Sistemas	02/10/2024

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Análisis y Desarrollo de Software.
- Código del Programa de Formación: 228118
- Nombre del Proyecto Formativo (si aplica): Integración de tecnologías en el desarrollo de software.
- Fase del Proyecto (si aplica): Análisis.
- Actividad de Proyecto Formativo (si aplica): Determinar las especificaciones funcionales del software y metodología a utilizar.
- Competencia: Utilizar herramientas informáticas de acuerdo con las necesidades de manejo de información.
- Resultados de Aprendizaje:
    - Aplicar funcionalidades de herramientas y servicios TIC, de acuerdo con manuales de uso, procedimientos establecidos y buenas prácticas.
    - Optimizar los resultados, de acuerdo con la verificación.
    - Evaluar los resultados, de acuerdo con los requerimientos.
    - Alistar herramientas de tecnologías de la información y la comunicación (TIC), de acuerdo con las necesidades de procesamiento de información y comunicación.
- Duración de la Guía de Aprendizaje (horas): 40.

2. PRESENTACIÓN

El manejo de herramientas informáticas es una habilidad esencial en el mundo actual, tanto en el ámbito académico como profesional. Las tecnologías digitales están profundamente integradas en nuestra vida cotidiana, y el dominio de diversas aplicaciones y programas informáticos se ha convertido en una necesidad para enfrentar los retos del mundo laboral y académico. No solo se trata de aprender a utilizar estas herramientas, sino de comprender su potencial para facilitar la resolución de problemas, optimizar el trabajo y mejorar la productividad.

El objetivo de esta presentación es motivar a los aprendices a explorar las herramientas informáticas con un enfoque no solo en el uso básico, sino en la comprensión más profunda de cómo estas pueden apoyar su aprendizaje y desarrollo autónomo. Es fundamental que cada aprendiz asuma un rol activo en su proceso de aprendizaje, reconociendo que las herramientas tecnológicas no son únicamente recursos prácticos, sino aliados estratégicos para su crecimiento personal y académico.

Para motivar actividades de aprendizaje, es importante partir de lo que los aprendices ya conocen. Muchos de ellos ya están familiarizados con ciertos programas o aplicaciones como procesadores de texto, hojas de cálculo o plataformas de presentaciones. Sin embargo, el reto no es solo utilizar estos programas de forma mecánica, sino llevar su conocimiento un paso más allá, explorando nuevas funcionalidades que les permitan automatizar tareas, gestionar proyectos de manera eficiente o colaborar de forma remota con otros aprendices. A través de esta presentación, se pretende construir puentes entre los conocimientos previos de los aprendices y los nuevos conceptos que aprenderán, mostrando cómo la integración de estas herramientas en su día a día puede hacer la diferencia en su rendimiento.

El trabajo autónomo es otro pilar fundamental en el uso de herramientas informáticas. A través de un enfoque basado en la autonomía, los aprendices podrán gestionar su tiempo de manera más eficiente, organizar sus tareas y realizar investigaciones con mayor profundidad. Aprender a utilizar un software no es simplemente seguir un tutorial, sino descubrir cómo esa herramienta puede ajustarse a las necesidades individuales de cada persona. Esto fomenta el aprendizaje autodirigido, donde los aprendices exploran las funcionalidades avanzadas de las aplicaciones, ajustándolas a sus propios proyectos académicos y personales. De esta manera, las herramientas informáticas se convierten en vehículos para el desarrollo de habilidades clave como la resolución de problemas, la gestión del tiempo y la creatividad.

Además, no se puede ignorar el valor del aprendizaje colaborativo en este contexto. Muchas herramientas informáticas actuales están diseñadas específicamente para potenciar el trabajo en equipo. Plataformas de colaboración como Google Drive, Microsoft Teams o herramientas de gestión de proyectos como Trello permiten a los estudiantes trabajar de manera sincrónica y asincrónica, intercambiando ideas, compartiendo recursos y colaborando en tiempo real, sin las limitaciones de la distancia o los horarios. Estas herramientas no solo facilitan la organización del trabajo en grupo, sino que también permiten desarrollar habilidades interpersonales clave como la comunicación, la negociación y la toma de decisiones en conjunto.

El manejo adecuado de las herramientas informáticas, por lo tanto, no solo responde a la necesidad de adquirir destrezas técnicas, sino también a la necesidad de fomentar el trabajo en equipo y la autogestión. Al aprender a utilizar estas herramientas, los aprendices no solo amplían sus competencias digitales, sino que también desarrollan una mentalidad colaborativa y autónoma, donde el intercambio de conocimientos y experiencias es crucial para el crecimiento conjunto.

Finalmente, es importante resaltar que el manejo de herramientas informáticas no debe verse como una competencia aislada, sino como una habilidad transversal que puede aplicarse en diferentes contextos académicos y profesionales. Ya sea en la elaboración de informes, el análisis de datos, la presentación de proyectos o la planificación de actividades, las herramientas tecnológicas ofrecen

infinitas posibilidades de mejorar y optimizar el trabajo. Por eso, es fundamental que los aprendices comprendan que, más allá de aprender un software específico, están adquiriendo habilidades que les serán útiles a lo largo de toda su vida académica y profesional.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de reflexión inicial:

Descripción de la actividad:

La actividad comienza con una breve introducción, explicando que las herramientas informáticas están presentes en casi todos los aspectos de nuestra vida, desde la forma en que nos comunicamos hasta cómo organizamos nuestro tiempo y realizamos tareas académicas o laborales. Se invita a los aprendices a reflexionar sobre la influencia que estas herramientas tienen en sus actividades cotidianas, y cómo impactan su productividad y aprendizaje.

Se les pide a los aprendices que, de manera individual, respondan a la siguiente pregunta: "¿Qué herramienta informática utilizas con más frecuencia en tu vida diaria y cómo crees que ha influido en la forma en que trabajas, estudias o te comunicas?".

Los aprendices tienen unos minutos para escribir sus reflexiones de forma breve. Pueden mencionar cualquier tipo de herramienta, ya sea un programa de software, una aplicación móvil o una plataforma de comunicación, y explicar cómo ha mejorado o cambiado su rutina diaria, su capacidad de organizarse o comunicarse con otros.

Después de escribir sus respuestas, se les pide a algunos voluntarios que compartan sus reflexiones con el grupo. Esta discusión permite a los aprendices identificar similitudes en sus experiencias, reconocer la omnipresencia de las herramientas digitales y generar un espacio para compartir cómo el uso de estas aplicaciones les ayuda a enfrentar distintos desafíos.

A través de esta reflexión inicial, los aprendices comenzarán a tomar conciencia de la importancia de las herramientas informáticas en su vida cotidiana y estarán más abiertos a explorar nuevas aplicaciones y software en el marco de su formación académica. Se puede cerrar la actividad resaltando que las herramientas tecnológicas no solo facilitan tareas, sino que son instrumentos poderosos para el aprendizaje autónomo, el trabajo colaborativo y el crecimiento personal.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Equipo de cómputo.

Material de apoyo: Documento en PDF sobre herramientas tecnológicas.

Duración de la actividad: 4 horas.

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Descripción de la actividad:

Se inicia la actividad planteando el siguiente reto a los aprendices: "¿Cuántas herramientas informáticas conoces y para qué las utilizas?". Esto servirá como punto de partida para que los aprendices reconozcan el amplio abanico de aplicaciones informáticas que usan en su vida diaria, tanto en el ámbito académico, como personal o profesional.

A continuación, los aprendices se organizan en grupos pequeños de tres o cuatro personas. Cada grupo debe elaborar un mapa de herramientas informáticas, dividiéndolo en diferentes categorías, como por ejemplo: procesamiento de textos, comunicación, gestión de proyectos, diseño gráfico, análisis de datos, entretenimiento, entre otras. En cada categoría, los aprendices deben listar las herramientas que conocen o han utilizado, así como describir brevemente su función principal.

Una vez que han terminado de crear sus mapas, cada grupo debe elegir una herramienta que consideren más relevante o interesante en cada categoría y explicar al resto de la clase cómo la utilizan y en qué contexto. Por ejemplo, un grupo podría hablar sobre Google Docs en la categoría de procesamiento de textos, Trello para la gestión de proyectos, o Canva en diseño gráfico.

Durante la discusión, se toma nota de las herramientas mencionadas y hace preguntas para profundizar en el nivel de conocimiento que tienen sobre cada una, indagando si los aprendices conocen funciones avanzadas o si solo las utilizan de manera básica. Esto permite identificar no solo las herramientas más

utilizadas, sino también las lagunas en el conocimiento y las áreas donde pueden mejorar o explorar nuevas aplicaciones.

Al finalizar la actividad, se recapitulan las principales herramientas mencionadas, y se puede introducir nuevas herramientas que complementen o mejoren las que ya conocen los aprendices. También se anima a los aprendices a reflexionar sobre cómo algunas de las herramientas mencionadas pueden ser útiles en su futuro académico o profesional, abriendo el espacio para relacionar los conocimientos previos con los nuevos aprendizajes que se abordarán en el curso.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Equipo de cómputo.

Material de apoyo: Documento en PDF sobre herramientas tecnológicas.

Duración de la actividad: 4 horas.

3.3 Actividades de apropiación:

Descripción de la actividad:

La actividad consiste en que los aprendices, organizados en pequeños grupos, desarrollen un proyecto en el que utilicen diversas herramientas informáticas para abordar un tema de interés relacionado con su área de estudio o un problema práctico de su entorno. El objetivo es que, a través de este proyecto, los aprendices se apropien de las herramientas exploradas en el curso, utilizando varias de ellas para completar diferentes fases del proyecto (investigación, planificación, producción y presentación).

Cada grupo debe elegir un tema o problema que deseen abordar. Pueden optar por algo relacionado con su ámbito académico (por ejemplo, desarrollar una campaña de concienciación sobre un tema de actualidad, organizar un evento o crear un informe detallado de una investigación) o un proyecto práctico (como la creación de un blog o la planificación de un viaje). A partir de aquí, deben utilizar una

herramienta de gestión de proyectos (como Trello o Asana) para distribuir las tareas entre los miembros del grupo, fijar plazos y organizar el flujo de trabajo.

En esta fase, los aprendices deberán utilizar herramientas como Google Drive para colaborar en la creación de documentos, compartiendo enlaces, imágenes y notas relevantes para la investigación del tema seleccionado. También pueden emplear herramientas específicas de recopilación de datos o investigación, como gestores de referencias bibliográficas (Mendeley o Zotero) o plataformas de análisis de datos básicos (Google Forms o Microsoft Excel), para analizar encuestas o resultados de investigaciones previas.

Dependiendo del proyecto seleccionado, los aprendices utilizarán diversas herramientas para producir el contenido necesario. Si el proyecto es un informe, pueden emplear un procesador de texto avanzado como Microsoft Word o Google Docs, aprovechando funciones de colaboración en tiempo real. Si se trata de un contenido audiovisual, pueden utilizar herramientas como Canva para diseño gráfico o una aplicación de edición de video como iMovie o Adobe Premiere.

Finalmente, los aprendices deberán presentar su proyecto utilizando una herramienta de presentación como Microsoft PowerPoint, Prezi o Google Slides, combinando las diferentes habilidades que han adquirido a lo largo de la actividad. La presentación debe ser clara, visualmente atractiva y mostrar cómo utilizaron las herramientas informáticas para abordar cada fase del proyecto. Pueden incluir gráficos, tablas, infografías o videos para hacer la exposición más dinámica.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:
Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Equipo de cómputo.
Material de apoyo: Documento en PDF sobre herramientas tecnológicas.
Evidencias de aprendizaje: Taller herramientas tecnológicas.

Instrumentos de evaluación: Prueba de conocimientos.

Duración de la actividad: 4 horas.

3.4 Actividades de Transferencia el Conocimiento:

Descripción de la actividad:

La actividad inicia con una introducción sobre la importancia de utilizar herramientas informáticas para resolver problemas en diversas áreas, como la educación, la salud, la comunidad o el medio ambiente. Se presenta a los aprendices una serie de retos o problemas reales que enfrentan en su entorno, invitándolos a pensar en cómo podrían abordarlos utilizando las herramientas informáticas aprendidas en clase.

Los aprendices se organizan en grupos de trabajo y eligen uno de los problemas propuestos o pueden identificar un desafío propio que quieran abordar. Ejemplos de problemas podrían incluir la falta de comunicación en un proyecto escolar, la organización de un evento comunitario, o la gestión del tiempo personal entre estudio y ocio.

Una vez que los grupos han seleccionado un problema, deben investigar y analizar la situación en profundidad. Esto implica recopilar información sobre el contexto, las partes interesadas y las posibles soluciones. Utilizando herramientas como Google Forms, pueden diseñar encuestas para obtener opiniones de otros aprendices o miembros de la comunidad, o emplear Google Docs para hacer un brainstorming de ideas.

Con la información recabada, cada grupo debe diseñar una solución digital que aborde el problema elegido. Esto puede incluir la creación de una aplicación sencilla, un sitio web, un documento colaborativo que explique cómo mejorar la situación, o una presentación visual que proponga cambios. Los aprendices tendrán la libertad de elegir las herramientas informáticas que consideren más adecuadas para su solución, como Canva para el diseño gráfico, Google Sites para crear un sitio web, o plataformas de programación sencillas como Scratch.

Cada grupo presenta su solución al resto de la clase, explicando el problema que eligieron, el proceso que siguieron para investigarlo y cómo su solución digital aborda el desafío. Durante las

presentaciones, los otros grupos pueden hacer preguntas o proporcionar retroalimentación constructiva, lo que fomenta el aprendizaje colaborativo y la reflexión crítica.

Finalmente, se lleva a cabo una reflexión grupal donde los aprendices discuten lo que aprendieron a lo largo de la actividad. Se les invita a pensar en cómo el conocimiento que adquirieron sobre las herramientas informáticas les permitió desarrollar una solución creativa y útil para un problema real. También se les pregunta cómo podrían aplicar este enfoque a otros desafíos en el futuro.

Ambiente requerido: Ambiente de formación.

Estrategias o técnicas didácticas activas:

Aprendizaje basado en experiencias, trabajo colaborativo y en grupo, refuerzo positivo y autoevaluación y reflexión.

Materiales de formación: Equipo de cómputo.

Material de apoyo: Documento en PDF sobre herramientas tecnológicas.

Evidencias de aprendizaje: Taller de aplicación de herramientas.

Instrumentos de evaluación: Prueba de conocimientos.

Duración de la actividad: 4 horas.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Análisis	Determinar las especificaciones funcionales del software y metodología a utilizar.	Herramientas informáticas.	Taller herramientas tecnológicas. Taller de aplicación de herramientas.	Identifica equipos tic, tipos de software y servicios de internet, de acuerdo con las	Prueba de conocimientos.

				Necesidades de uso. Compara equipos tic, tipos de software y servicios de internet, de acuerdo con las características. Escoge equipos tic, tipos de software y servicios de internet, de acuerdo con las necesidades de procesamiento de información y características. Maneja procesador de texto, hoja de cálculo, software para presentaciones, diagramación, bases de datos y software específico, de acuerdo con las funcionalidades de cada	

				programa y las necesidades de manejo de la información.	

5. GLOSARIO DE TÉRMINOS

Herramientas Informáticas: Aplicaciones y programas diseñados para realizar tareas específicas en entornos digitales, como procesamiento de texto, análisis de datos, diseño gráfico y gestión de proyectos.

Procesador de Texto: Software que permite crear, editar y formatear documentos de texto, facilitando la escritura y presentación de información. Ejemplos: Microsoft Word, Google Docs.

Hojas de Cálculo: Programas que permiten organizar, analizar y visualizar datos en formato tabular. Se utilizan comúnmente para realizar cálculos y generar gráficos. Ejemplos: Microsoft Excel, Google Sheets.

Presentaciones: Herramientas que permiten crear y mostrar presentaciones visuales compuestas de diapositivas, utilizando texto, imágenes y gráficos. Ejemplos: Microsoft PowerPoint, Google Slides.

Gestión de Proyectos: Proceso de planificar, organizar y coordinar recursos para alcanzar objetivos específicos dentro de un plazo determinado. Herramientas como Trello y Asana son ejemplos de software que facilitan esta gestión.

Colaboración en Tiempo Real: Capacidad de múltiples usuarios de trabajar simultáneamente en un documento o proyecto, permitiendo la edición y comunicación instantánea. Ejemplo: Google Docs permite que varios usuarios colaboren en un mismo documento en tiempo real.

Infografía: Representación visual de información o datos, diseñada para presentar información de manera clara y atractiva. Herramientas como Canva facilitan la creación de infografías.

Análisis de Datos: Proceso de inspeccionar, limpiar y modelar datos con el objetivo de descubrir información útil, extraer conclusiones y apoyar la toma de decisiones. Herramientas como Microsoft Excel y Google Analytics son utilizadas para este propósito.

Encuesta: Herramienta utilizada para recopilar datos o información de un grupo de personas sobre un tema específico, generalmente a través de preguntas estructuradas. Google Forms es una herramienta popular para crear encuestas.

Desarrollo de Software: Proceso de diseño, creación y mantenimiento de aplicaciones y programas informáticos. Incluye la escritura de código, pruebas y depuración.

Aplicación Móvil: Programa diseñado para ser ejecutado en dispositivos móviles, como teléfonos inteligentes o tabletas, que proporciona funcionalidades específicas a los usuarios.

Nube: Tecnología que permite almacenar y acceder a datos y aplicaciones a través de internet, en lugar de hacerlo localmente en un dispositivo. Ejemplos de servicios en la nube son Google Drive y Dropbox.

Creatividad Digital: Habilidad de utilizar herramientas digitales para generar ideas originales y presentar información de manera innovadora. Se manifiesta en la producción de contenido visual, multimedia y textual.

Autoevaluación: Proceso mediante el cual un individuo reflexiona sobre su propio aprendizaje, identificando fortalezas y áreas de mejora en sus habilidades y conocimientos.

Pensamiento Crítico: Habilidad para analizar y evaluar información y argumentos de manera lógica y razonada, considerando diferentes perspectivas antes de llegar a una conclusión.

Trabajo Colaborativo: Proceso en el que un grupo de personas trabaja de manera conjunta para alcanzar un objetivo común, utilizando habilidades y conocimientos diversos.

Solución Digital: Resultado de aplicar herramientas informáticas para resolver un problema específico o mejorar un proceso. Puede incluir aplicaciones, sistemas o métodos digitales.

6. REFERENTES BILBIOGRÁFICOS

Aguirre, A. (2015). Las TIC en la educación: Una mirada crítica.

Castañeda, L., & Adell, J. (2013). Educación y TIC: El papel de las tecnologías en la enseñanza y el aprendizaje.

García, A. (2016). Didáctica de las TIC: Estrategias y herramientas para la educación.

Gómez, R., & Martínez, M. (2014). Las TIC en la educación: Herramientas para el aprendizaje.

Hernández, A. (2018). Educación, tecnología y cambio: Un enfoque crítico.

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	Raúl Gómez	Instructor	Red de Informática y Sistemas	02/10/2024

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					