RELACIÓN 1 DIAGRAMAS UML

Ejercicios DIAGRAMAS DE CLASES

1. Imagina que estás diseñando un sistema para una tienda en línea. La tienda vende productos de diferentes categorías y permite a los clientes crear cuentas para realizar pedidos.

Crea un diagrama de clases que incluya las clases Producto, Categoría, Cliente y Pedido y muestra las relaciones entre ellas.

- La clase **Producto** tiene los siguientes atributos: nombre, precio y descripción, y un método mostrarInfo() para mostrar la información del producto.
- La clase **Categoría** tiene un atributo nombre y métodos para agregar y eliminar productos de la categoría.
- La clase **Pedido** tiene los atributos fecha y total, y métodos para agregar y eliminar productos del pedido y calcular el total.
- La clase **Cliente** tiene los atributos nombre, dirección, usuario y clave, y métodos para hacer pedidos y ver el historial de pedidos.

a. Representa las clases
b. ¿Cuáles podrían ser los parámetros y los tipos de retorno de los métodos?
c. Indica las relaciones entre las clases y de qué tipo podrían ser.
d. ¿Cuáles son las cardinalidades de las relaciones anteriores?
e. Indica posibles atributos de enlace para las relaciones planteadas.

SOLUCIÓN:

NOTA: Dentro de los diagramas estructurales, y de todos los UML en general, el diagrama de clases es el más importante porque representa los elementos estáticos del sistema, sus atributos y comportamientos, y como se relacionan entre ellos. Contiene las clases del dominio del problema, y a partir de éste se obtendrán las clases que formarán después el programa informático que dará solución al problema.

a. Según la teoría disponible en la Unidad 5, tenemos la siguiente definición de clase:
Clases: abstracciones del dominio del sistema que representan elementos del mismo mediante una serie de características, que llamaremos atributos, y su comportamiento, que serán métodos. Los atributos y métodos tendrán una visibilidad que determinará quien puede acceder al atributo o método. Por ejemplo, una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará.

Atributos: Forman la parte estática de la clase. Son un conjunto de variables para las que es preciso definir:

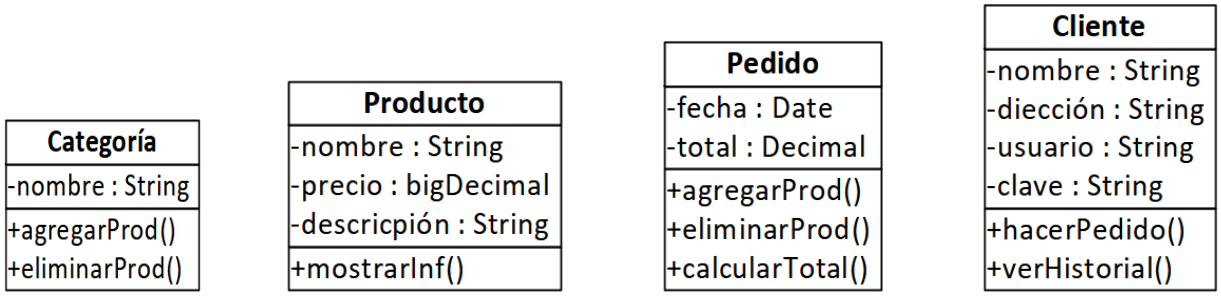
- Su **nombre**.

- Su **tipo**, puede ser un tipo simple, que coincidirá con el tipo de dato que se seleccione en el lenguaje de programación final a usar, o compuesto, pudiendo incluir otra clase.

Además, se pueden indicar otros datos como un valor inicial o su visibilidad. La visibilidad de un atributo se puede definir como:

- **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
- **Privado**: Sólo se pueden acceder desde operaciones de la clase.
- **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
- **Paquete**: Se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación es Java.

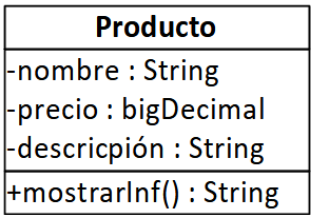
Representemos las clases de nuestro enunciado:



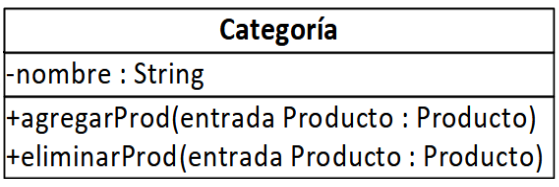
b. Según la teoría de la Unidad 5:
   Métodos: Representan la funcionalidad de la clase, es decir, qué puede hacer. Para definir un método hay que indicar como mínimo su nombre, parámetros, el tipo que devuelve y su visibilidad. También se debe incluir una descripción del método que aparecerá en la documentación que se genere del proyecto.

Los parámetros y tipos de retorno de los métodos en el diagrama de clases que hemos representado pueden variar según las necesidades específicas del sistema que se está diseñando. Una posible solución para los parámetros y tipos de retorno de los métodos en cada clase podría ser la siguiente:

Clase Producto:
El método `mostrarInfo()` no tiene parámetros y podría devolver una cadena que contenga la información del producto.



Clase Categoría:
El método agregarProd() podría tener un parámetro *producto* de tipo 'producto' y no devolver nada.



El método eliminarProd() podría tener un parámetro *producto* de tipo 'producto' y no devolver nada.

Clase Pedido:

El método agregarProd() podría tener un parámetro *producto* de tipo 'producto' y no devolver nada.
El método eliminarProd() podría tener un parámetro *producto* de tipo 'producto' y no devolver nada.
El método calcularTotal() no tiene parámetros y podría devolver un número que represente el total del pedido.

Pedido
-fecha : Date -total : Decimal
+agregarProd(entrada Producto : Producto) +eliminarProd(entrada Producto : Producto) +calcularTotal() : bigDecimal

Clase Cliente:

El método hacerPedido() podría tener un parámetro *pedido* de tipo 'pedido' y no devolver nada.
El método verHistorial() no tiene parámetros y podría devolver una lista de objetos de tipo 'pedido' que representen el historial de pedidos del cliente.

Cliente
-nombre : String -diección : String -usuario : String -clave : String
+hacerPedido(entrada Pedido : Pedido) +verHistorial() : String

c. Relaciones entre las clases:

Una **relación** es una conexión entre dos clases que incluimos en el diagrama cuando aparece algún tipo de relación entre ellas en el dominio del problema.

Se representan como una línea continua. Los mensajes "navegan" por las relaciones entre clases, es decir, los mensajes se envían entre objetos de clases relacionadas, normalmente en ambas direcciones, aunque a veces la definición del problema hace necesario que se navegue en una sola dirección, entonces la línea finaliza en punta de flecha.

Para nuestro ejemplo:

- La clase **Producto** está relacionada **con** la clase **Categoría** ya que un producto pertenece a una categoría.

- La clase **Pedido** está relacionada **con** la clase **Producto** ya que un pedido puede contener varios productos.

- La clase **Cliente** está relacionada **con** la clase **Pedido** ya que un cliente puede hacer varios pedidos.

Las relaciones pueden ser:

- De asociación: una relación de asociación entre dos clases indica que existe una conexión entre ellas. Esta conexión puede ser bidireccional o unidireccional y puede tener diferentes multiplicidades (cardinalidades) para indicar cuántas instancias de una clase están relacionadas con cuántas instancias de la otra clase.
- De herencia; La **herencia** es una propiedad que permite a los objetos ser construidos a partir de otros objetos, es decir, la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados. El objetivo principal de la herencia es la *reutilización*, poder utilizar código desarrollado con anterioridad.
- De agregación; La **agregación** es una asociación binaria que representa una relación todo-parte (pertenece a, tiene un, es parte de). Los elementos parte pueden existir sin el elemento contenedor y no son propiedad suya. Por ejemplo, un centro comercial tiene clientes o un equipo tiene unos miembros. El tiempo de vida de los objetos no tiene porqué coincidir.
- De composición; La **composición** es una agregación fuerte en la que una instancia 'parte' está relacionada, como máximo, con una instancia 'todo' en un momento dado, de forma que cuando un objeto 'todo' es eliminado, también son eliminados sus objetos 'parte'. Por ejemplo: un rectángulo tiene cuatro vértices, un centro comercial está organizado mediante un conjunto de secciones de venta...

En nuestro ejemplo esta es una posible solución para los tipos de relaciones entre las clases:

- La relación entre la clase Producto y la clase Categoría podría ser una relación de **asociación**, lo que significa que un producto está asociado con una categoría.
- La relación entre la clase Pedido y la clase Producto podría ser una relación de **agregación**, lo que significa que un pedido está compuesto por varios productos. (La representamos de asociación)
- La relación entre la clase Cliente y la clase Pedido podría ser una relación de **composición**, lo que significa que un cliente es responsable de sus pedidos. (La representamos de asociación)

d. Cardinalidad

Un concepto muy importante es la **cardinalidad de una relación**, representa cuantos objetos de una clase se van a relacionar con objetos de otra clase. En una relación hay dos cardinalidades, una para cada extremo de la relación y pueden tener los siguientes valores:

Significado de las cardinalidades.	
Cardinalidad	**Significado**
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Varios
0.. *	Cero o varios
1.. *	Uno o varios (al menos uno)

Las **cardinalidades** de las relaciones en el diagrama de clases del ejercicio pueden variar según las necesidades específicas del sistema que se está diseñando. Una posible solución para las cardinalidades de las relaciones entre las clases:

- La relación entre la clase Producto y la clase Categoría podría ser de **"muchos a uno",** lo que significa que un producto pertenece a una categoría y una categoría puede tener muchos productos.

- La relación entre la clase Pedido y la clase Producto podría ser de **"muchos a muchos",** lo que significa que un pedido puede contener varios productos y un producto puede estar en varios pedidos.

- La relación entre la clase Cliente y la clase Pedido podría ser de **"uno a muchos",** lo que significa que un cliente puede hacer varios pedidos y un pedido pertenece a un cliente.

e. Atributos de enlace

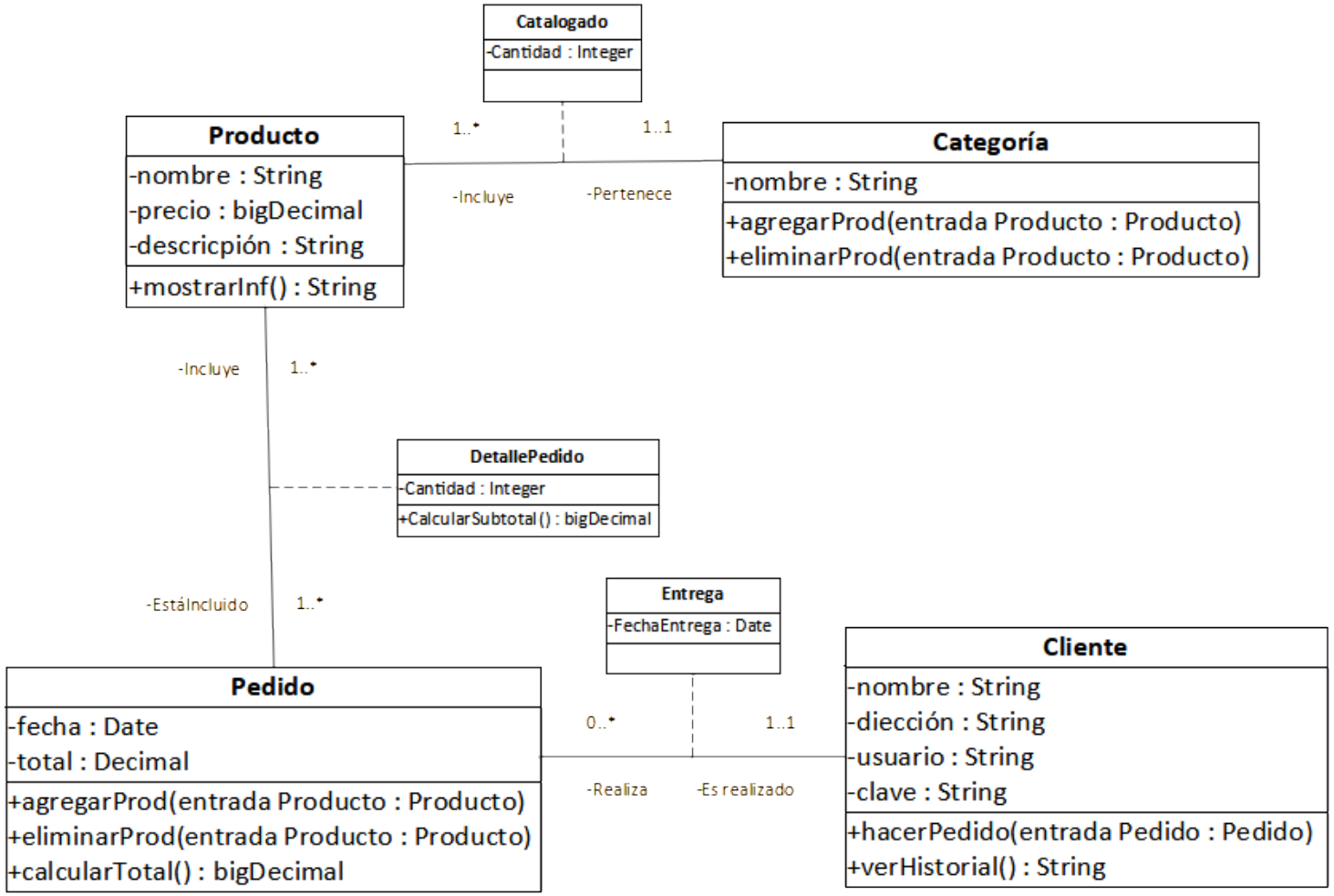
Un atributo de enlace es un atributo que se utiliza para representar información adicional sobre una relación entre dos clases en un diagrama de clases. Los atributos de enlace pueden variar según las necesidades específicas del sistema que se está diseñando. Algunos ejemplos de posibles atributos de enlace para las relaciones en el diagrama de clases anterior podrían ser los siguientes:

- La relación entre la clase Producto y la clase Categoría podría tener un atributo de enlace llamado **cantidad** para representar la cantidad de productos de una categoría específica que están disponibles en la tienda.

- La relación entre la clase Pedido y la clase Producto podría tener un atributo de enlace llamado **cantidad** para representar la cantidad de un producto específico que se ha agregado a un pedido. (En este ejemplo, hemos agregado una clase asociativa llamada DetallePedido entre las clases Pedido y Producto. Esta clase asociativa tiene un atributo llamado cantidad que representa el atributo de enlace entre las dos clases.

También hemos agregado un método llamado **calcularSubtotal()** para calcular el subtotal del detalle del pedido.)

- La relación entre la clase Cliente y la clase Pedido podría tener un atributo de enlace llamado **fechaEntrega** para representar la fecha en que se entregará un pedido a un cliente.


Catalogado
-Cantidad : Integer
Producto
-nombre : String
-precio : bigDecimal
-descricpión : String
+mostrarInf() : String
1..*
1..1
-Incluye
-Pertenece
Categoría
-nombre : String
+agregarProd(entrada Producto : Producto)
+eliminarProd(entrada Producto : Producto)
-Incluye
1..*
DetallePedido
-Cantidad : Integer
+CalcularSubtotal() : bigDecimal
-EstáIncluido
1..*
Entrega
-FechaEntrega : Date
Pedido
-fecha : Date
-total : Decimal
+agregarProd(entrada Producto : Producto)
+eliminarProd(entrada Producto : Producto)
+calcularTotal() : bigDecimal
0..*
1..1
-Realiza
-Es realizado
Cliente
-nombre : String
-diección : String
-usuario : String
-clave : String
+hacerPedido(entrada Pedido : Pedido)
+verHistorial() : String