**Materia:** Desarrollo Back End – 1° Trabajo Práctico.

**Profesor:** Ing. Tulio Ruesjas Martin.

# Fecha de Entrega : 05 de setiembre de 2023 Nombre y Apellido del Alumno: BOUSO GONZALO

**Temario:**

1. ¿Defina que es una clase y atributo en OOP(Object Oriented Programming)?

Desarrollo

**Clase**

En la Programación Orientada a Objetos (OOP, por sus siglas en inglés), una **clase** es un concepto fundamental que define un tipo de objeto. Se puede pensar en una clase como una plantilla o un molde que describe las propiedades y comportamientos que los objetos de esa clase tendrán.

* **Propiedades**: Son las características o datos que los objetos de la clase tendrán. Estos se definen mediante atributos.
* **Comportamientos**: Son las acciones o métodos que los objetos pueden realizar. Estos se definen mediante métodos.

Por ejemplo, si tienes una clase llamada Coche, esta clase podría definir que todos los coches tienen ciertas propiedades (como marca, modelo y color) y ciertos comportamientos (como arrancar() y frenar()).

**Atributo**

Un **atributo** es una variable que pertenece a una clase y que define una característica o propiedad de los objetos creados a partir de esa clase. Los atributos almacenan datos sobre el estado de un objeto.

En el ejemplo anterior, marca, modelo y color son atributos de la clase Coche. Cada objeto de la clase Coche tendrá su propia copia de estos atributos con valores específicos.

Los atributos pueden ser:

* **Atributos de instancia**: Son específicos para cada instancia (objeto) de la clase. En el ejemplo, marca, modelo y color son atributos de instancia porque cada coche puede tener una marca, modelo y color diferentes.
* **Atributos de clase**: Son compartidos por todas las instancias de la clase. Estos atributos son comunes a todas las instancias y se definen directamente dentro de la clase pero fuera de cualquier método.

1. Se desea hacer el modelado de clases de una compañía aérea en una base de datos relacional. La compañía aérea tiene tres recursos principales: aviones, pilotos y miembros de tripulación. De cada piloto se desea conocer su código, nombre y horas de vuelo. De los miembros de tripulación sólo mantendremos su código y nombre. Todos ellos (pilotos y miembros) tienen una base a la que regresan después de los vuelos de una jornada. Un vuelo que va desde un origen a un destino y a una hora determinada, tiene un número de vuelo (por ejemplo, el vuelo de Tucumán a Buenos Aires de las 13:50 es el vuelo IB-8830). De cada vuelo que se va a realizar durante los próximos tres meses, así como de los vuelos que ya se han realizado, se desea saber el avión en que se va a hacer o en el que se ha hecho, el piloto y cada uno de los miembros de la tripulación. Cada avión tiene un código, es de un tipo (por ejemplo, BOEING-747) y tiene una base donde es sometido a las revisiones periódicas de mantenimiento.

Solo se deben identificar las clases con sus atributos.

1. ¿Cuáles son las características de una clase?

DESARROLLO

**Nombre de la Clase**

* **Descripción**: El nombre de la clase es una etiqueta que identifica y diferencia la clase en el código. Debe ser significativo y reflejar el propósito de la clase.
* **Ejemplo**: Coche, Usuario, Factura.

**2. Atributos**

* **Descripción**: Los atributos son variables que almacenan datos o estado de un objeto. Estos definen las características del objeto.
* **Ejemplo**: En una clase Coche, los atributos podrían ser marca, modelo, y color.

**3. Métodos**

* **Descripción**: Los métodos son funciones definidas dentro de la clase que describen los comportamientos que los objetos de la clase pueden realizar. Los métodos pueden operar sobre los atributos de la clase y realizar acciones específicas.
* **Ejemplo**: En una clase Coche, los métodos podrían ser arrancar(), frenar(), y acelerar().

1. **Constructor**.

* **Descripción**: El constructor es un método especial que se llama automáticamente cuando se crea una nueva instancia de la clase. Su propósito principal es inicializar los atributos del objeto.
* **Ejemplo**: En Python, el constructor se define con el método \_\_init\_\_().

1. **Encapsulamiento**

* **Descripción**: El encapsulamiento es el principio de ocultar los detalles internos de la implementación de una clase y exponer sólo lo necesario. Se logra mediante la definición de métodos públicos y privados. Los atributos privados no pueden ser accedidos directamente desde fuera de la clase.
* **Ejemplo**: En Python, los atributos que comienzan con un guion bajo (\_) son considerados privados, aunque no existe un mecanismo estricto para la encapsulación.

### **. Herencia**

* **Descripción**: La herencia permite crear una nueva clase basada en una clase existente. La nueva clase (subclase) hereda los atributos y métodos de la clase base (superclase) y puede agregar o modificar comportamientos.
* **Ejemplo**: Si Coche es la clase base, una clase Deportivo podría heredar de Coche y añadir características específicas.

### **Polimorfismo**

* **Descripción**: El polimorfismo permite que diferentes clases implementen métodos con el mismo nombre de manera diferente. Esto significa que el mismo método puede comportarse de manera distinta según el objeto que lo invoca.
* **Ejemplo**: Si varias clases, como Coche y Motocicleta, tienen un método arrancar(), cada clase puede implementar su propia versión del método.

### **Abstracción**

* **Descripción**: La abstracción es el proceso de simplificar la complejidad al enfocarse en los aspectos esenciales de una clase y ocultar los detalles no necesarios. Las clases abstractas pueden proporcionar una interfaz común para sus subclases, que deben implementar los métodos definidos en la clase abstracta.
* **Ejemplo**: En Python, las clases abstractas se definen usando el módulo abc.

1. ¿Qué es una interface? ¿Cual es la diferencia entre una clase y una interface?

DESARROLLO

Una **interfaz (interface)** es una estructura que define un contrato o conjunto de métodos y propiedades que una clase debe implementar. Es un modelo que asegura que cualquier clase que use esa interfaz debe tener ciertos métodos o propiedades definidos, pero no proporciona la implementación concreta de ellos. Las interfaces permiten que las clases compartan la misma estructura, independientemente de cómo implementen su funcionalidad.

**Diferencias entre una clase y una interfaz**:

1. **Definición de comportamiento vs. implementación**:
   * **Interfaz**: Solo define qué métodos y propiedades deben existir, sin detallar cómo se implementan. Es como un plan o guía.
   * **Clase**: Proporciona tanto la estructura como la implementación de métodos y propiedades.
2. **Herencia**:
   * **Interfaz**: Una clase puede implementar varias interfaces, permitiendo una forma de herencia múltiple. Sin embargo, la interfaz solo asegura que la clase tenga ciertas características.
   * **Clase**: Generalmente, una clase solo puede heredar de otra clase (herencia simple), lo que significa que hereda tanto la estructura como la implementación de esa clase.
3. **Instanciación**:
   * **Interfaz**: No se puede crear una instancia de una interfaz directamente. Es simplemente un contrato que obliga a una clase a implementar ciertos métodos.
   * **Clase**: Sí se puede instanciar una clase y crear objetos a partir de ella, ya que contiene la lógica para trabajar con los datos.
4. **Propósito**:
   * **Interfaz**: Facilita la interoperabilidad entre clases y asegura que diferentes clases puedan trabajar juntas si implementan la misma interfaz.
   * **Clase**: Sirve como una plantilla completa que define tanto la estructura de datos como el comportamiento de los objetos creados a partir de ella.

Las interfaces son útiles para definir reglas consistentes, mientras que las clases son necesarias para implementar esa lógica.

1. ¿Cuáles son las relaciones que existen entre clases?¿Cual es el concepto de cada una de esas relaciones?

DESARROLLO

Existen varias **relaciones entre clases** que se utilizan para modelar cómo interactúan entre sí en la programación orientada a objetos. Cada relación tiene un propósito y un concepto diferente. Estas son las principales relaciones:

### 1. **Herencia (Inheritance)**:

* **Concepto**: La herencia permite que una clase (subclase o clase derivada) herede atributos y métodos de otra clase (superclase o clase base). Esto permite reutilizar código y establecer una jerarquía entre clases.
* **Propósito**: Facilita la extensión de la funcionalidad de una clase existente sin necesidad de reescribir el código.
* **Ejemplo conceptual**: Un "Vehículo" podría ser una clase base y "Coche" y "Moto" serían subclases que heredan sus características básicas.

### 2. **Composición (Composition)**:

* **Concepto**: La composición es una relación "tiene un", donde una clase contiene instancias de otras clases como atributos. Es una relación más fuerte, ya que los objetos contenidos dependen completamente del objeto que los contiene.
* **Propósito**: Crear clases complejas utilizando clases más pequeñas y especializadas, lo que promueve la modularidad.
* **Ejemplo conceptual**: Un "Coche" tiene un "Motor". Si el coche deja de existir, el motor también dejará de existir.

### 3. **Agregación (Aggregation)**:

* **Concepto**: Es similar a la composición, pero más débil. En la agregación, una clase contiene instancias de otras clases, pero las instancias pueden existir independientemente del objeto que las contiene.
* **Propósito**: Modelar relaciones "tiene un", donde los componentes pueden tener una vida útil independiente.
* **Ejemplo conceptual**: Una "Universidad" puede tener varios "Estudiantes", pero los estudiantes no dejan de existir si la universidad se cierra.

### 4. **Asociación (Association)**:

* **Concepto**: Es una relación entre clases en la que una clase usa o interactúa con otra, pero sin que una posea a la otra. Puede ser unidireccional (una clase conoce a la otra) o bidireccional (ambas clases se conocen).
* **Propósito**: Representar una dependencia o interacción entre dos clases.
* **Ejemplo conceptual**: Un "Profesor" y un "Curso" están asociados porque el profesor enseña el curso, pero ninguno depende estructuralmente del otro.

### 5. **Dependencia (Dependency)**:

* **Concepto**: Es una relación débil en la que una clase depende de otra para realizar una función o tarea específica. Generalmente, es temporal, y una clase usa los servicios o datos de otra clase en un momento dado.
* **Propósito**: Representar el uso de una clase por otra, sin que exista una conexión estructural duradera.
* **Ejemplo conceptual**: Una "Factura" podría depender de una "Calculadora de Impuestos" para calcular el monto total, pero no la contiene de manera permanente.

### Resumen de relaciones:

* **Herencia**: Relación "es un".
* **Composición**: Relación "es parte de" o "tiene un" (fuerte).
* **Agregación**: Relación "tiene un" (débil).
* **Asociación**: Relación de interacción o colaboración.
* **Dependencia**: Relación temporal, una clase usa los servicios de otra.

1. ¿Cuáles son las diferencias entre Agregación y Composición?

DESARROLLO

Las diferencias entre **Agregación** y **Composición** radican en la naturaleza de la relación entre los objetos y el nivel de dependencia entre ellos. A continuación, se detallan las principales diferencias:

### 1. **Relación de propiedad**:

* **Agregación**: Es una relación más débil en la que una clase contiene instancias de otras clases, pero los objetos contenidos pueden existir independientemente del contenedor. La relación es del tipo "tiene un".
* **Composición**: Es una relación más fuerte en la que una clase contiene otras clases como parte de su estructura interna. Los objetos contenidos **no** pueden existir independientemente del contenedor. La relación también es del tipo "tiene un", pero con una dependencia más estricta.

### 2. **Vida útil de los objetos**:

* **Agregación**: Los objetos agregados pueden existir sin la clase que los contiene. Es decir, si la clase contenedora deja de existir, los objetos agregados pueden seguir existiendo de forma independiente.
* **Composición**: Los objetos que forman parte de una composición dependen completamente del objeto contenedor. Si el contenedor desaparece, los objetos que forman parte de él también se destruyen.

### 3. **Nivel de dependencia**:

* **Agregación**: Es una relación de asociación más débil. La clase contenedora conoce o utiliza a los objetos, pero no depende fuertemente de ellos. Estos objetos pueden existir sin la clase que los contiene.
* **Composición**: Es una relación de asociación más fuerte, en la que los objetos dependientes no pueden existir por sí mismos fuera del contenedor. Los objetos son inseparables del contenedor.

### 4. **Propiedad de los objetos**:

* **Agregación**: Los objetos agregados suelen tener su propia identidad y ciclo de vida. Pueden pertenecer a más de un objeto contenedor.
* **Composición**: Los objetos compuestos son parte integral del contenedor y no pueden ser compartidos con otros objetos fuera de su contexto.

### 5. **Ejemplos conceptuales**:

* **Agregación**: Una "Universidad" puede tener varios "Estudiantes". Si la universidad deja de existir, los estudiantes pueden seguir existiendo y pueden estar asociados con otras universidades.
* **Composición**: Un "Coche" tiene un "Motor". Si el coche deja de existir, el motor, como parte del coche, también deja de existir.

### Resumen:

* **Agregación**: Relación "tiene un" donde los objetos agregados pueden existir de forma independiente del contenedor.
* **Composición**: Relación "tiene un" donde los objetos dependientes no pueden existir sin el contenedor y forman parte integral de él.

1. ¿Qué es un diagrama de clases?

DESARROLLO

Un **diagrama de clases** es un tipo de diagrama utilizado en la **modelación de sistemas orientados a objetos**, que forma parte del lenguaje de modelado unificado (UML, por sus siglas en inglés). Representa la estructura estática de un sistema mostrando las **clases**, sus **atributos**, **métodos** (operaciones), y las **relaciones** entre ellas.

### Componentes principales de un diagrama de clases:

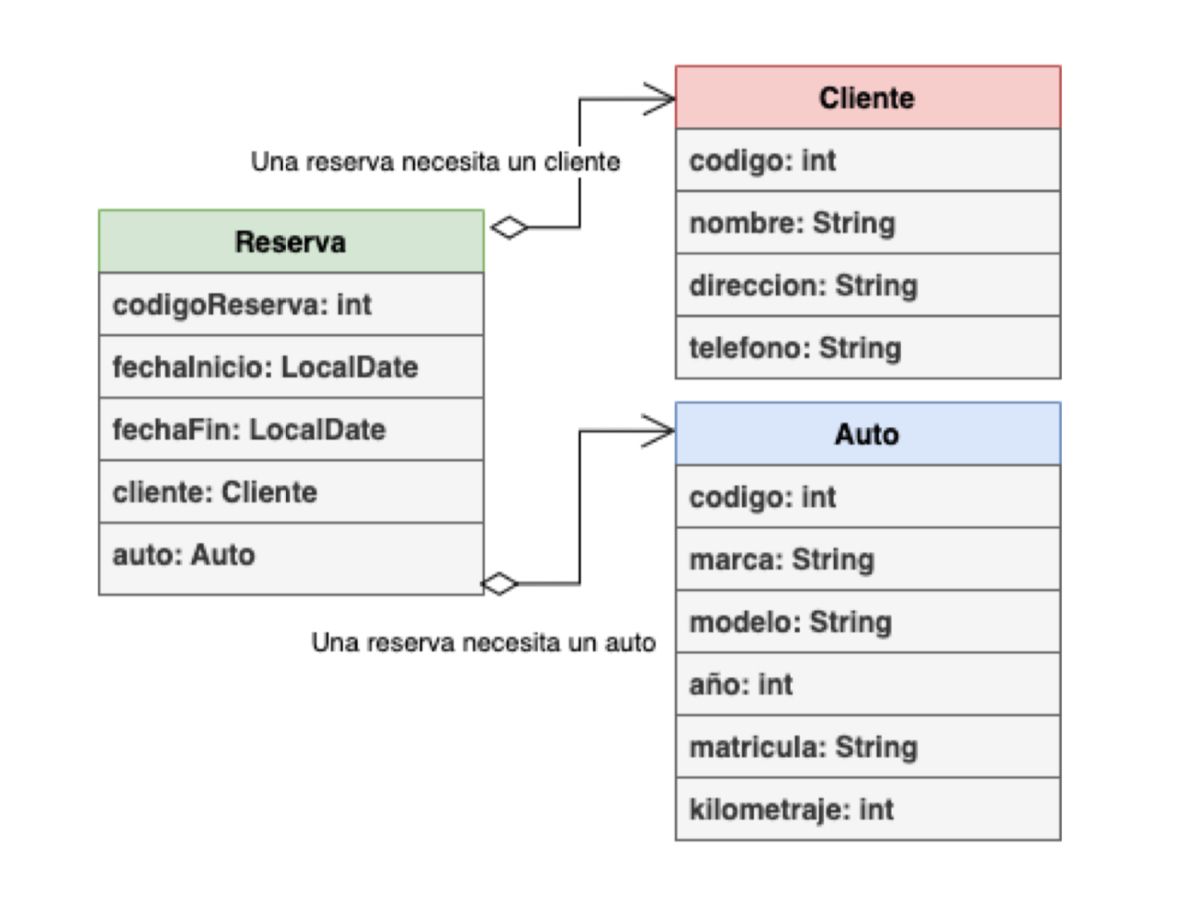
1. **Clases**:
   * Representadas como rectángulos divididos en tres secciones:
     + El nombre de la clase.
     + Sus atributos o propiedades (características).
     + Sus métodos u operaciones (comportamientos).
2. **Atributos**:
   * Son las propiedades o datos que pertenecen a la clase, definidos con su nombre y tipo.
   * Ejemplo: nombre: String, edad: Int.
3. **Métodos**:
   * Son las funciones o comportamientos que una clase puede realizar.
   * Ejemplo: calcularSalario(), imprimirDetalles().
4. **Relaciones entre clases**:
   * **Herencia (Generalización)**: Representa una relación "es un" donde una clase hija hereda las características de una clase padre.
   * **Asociación**: Muestra cómo las clases están conectadas o interactúan entre sí.
   * **Composición**: Indica que una clase contiene otra clase como parte integral de su estructura (representado con un rombo negro).
   * **Agregación**: Relación en la que una clase contiene otra, pero esta puede existir independientemente (representado con un rombo vacío).
   * **Dependencia**: Muestra que una clase depende de otra temporalmente para realizar una función.

### Propósito de un diagrama de clases:

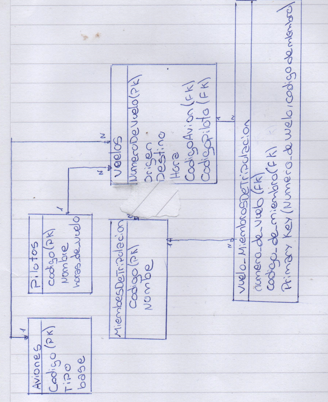
* **Visualizar la estructura del sistema**: Permite ver cómo están organizadas las clases y cómo interactúan entre sí.
* **Diseño del sistema**: Ayuda a los desarrolladores a planificar la arquitectura antes de escribir código, asegurando que las relaciones entre las clases sean claras.
* **Documentación**: Proporciona una representación gráfica que facilita la comprensión del sistema a lo largo de su ciclo de vida.

1. Elabore un diagrama de clases para una agencia de alquiler de autos.

DESARROLLO



DESARROLLO PUNTO 2



# Formato del Trabajo:

1. El trabajo deberá ser entregado en formato PDF.
2. Utiliza una estructura clara y coherente para cada sección del trabajo.
3. Cita adecuadamente las fuentes utilizadas para respaldar tus respuestas.
4. El trabajo debe ser enviado a: [tulio.ruesjas@unsta.edu.ar](mailto:tulio.ruesjas@unsta.edu.ar) hasta las 14 horas del día 05 de setiembre del 2024. TP enviado después de esa hora será rechazado.

# Firma Alumno Cantidad de Hojas(Incluida Enunciado)

|  |  |
| --- | --- |
| Nota | Firma Profesor |
|  |  |