El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

CICLO 2

EJE TEMÁTICO 2

PROGRAMACIÓN ORIENTADA A OBJETOS Y UML

Universidad Industrial de Santander

UIS

MisionTIC2022

Programación orientada a objetos y UML

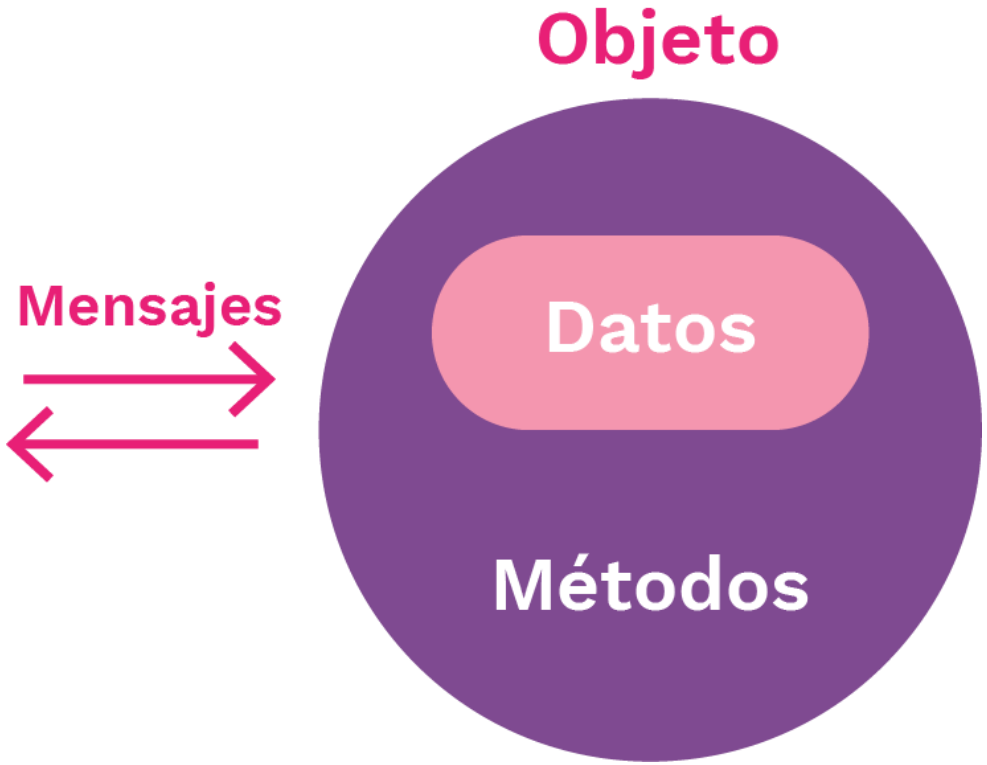
Ruta de aprendizaje

Programación orientada a objetos

En la programación orientada a objetos, se busca representar objetos con determinadas características, los cuales son creados a partir del uso de clases. Una clase en programación es una plantilla que contiene una estructura de datos que se identifica a través de un nombre, el cual se recomienda que vaya acorde a lo que ella representa. Dentro de esta clase van todas las características y funcionalidades que son propias del objeto que queremos representar con dicha clase.

Por medio de las clases es que se puede, en programación, construir objetos y un conjunto de estos objetos que se comunican entre sí a través de mensajes, que son los que conforman un programa. La programación orientada a objetos es la base de la programación en Java, y dentro de esta, el uso de clases y objetos es de gran importancia para la construcción de programas.

Mecanismos básicos de la programación orientada a objetos son:

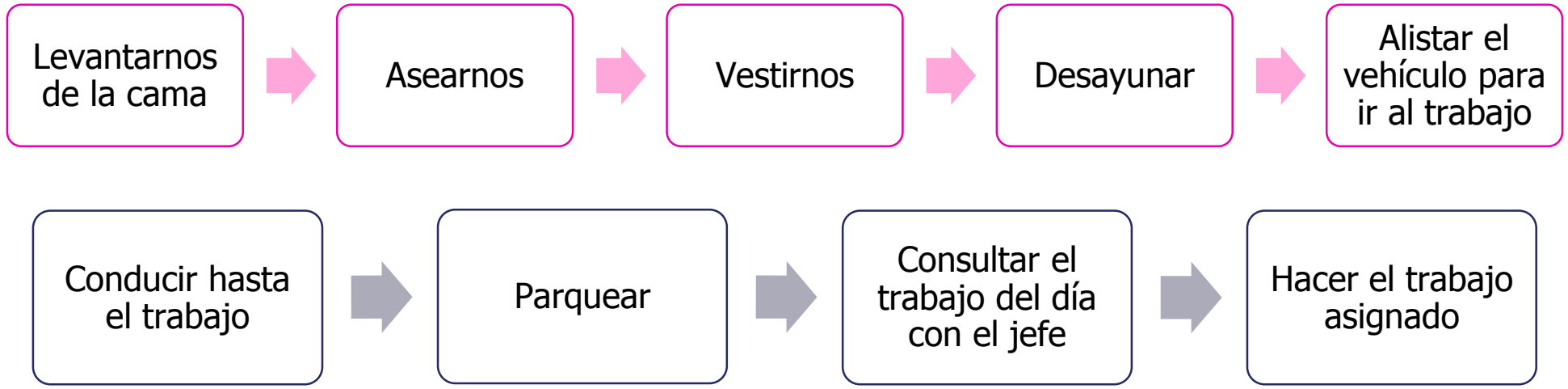


- ***Objetos***: Son los componentes principales de un programa orientado a objetos. Estos se pueden ver como datos abstractos compuestos de datos más simples y de procedimientos o funciones para la manipulación de estos.

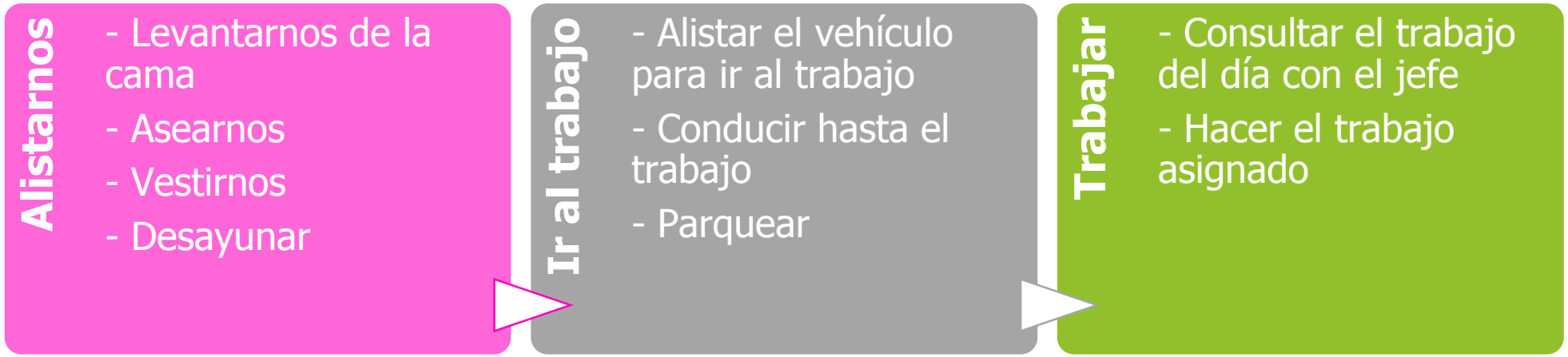
- ***Mensajes***: Un programa orientado a objetos al ejecutarse, cada uno de los objetos que lo componen, reciben, interpretan y responden a mensajes que envían otros objetos. Los mensajes son la manera a través de la cual se comunican los objetos.

- ***Métodos***: son implementaciones o funciones pertenecientes a una clase determinada, los cuales son accesibles mediante los objetos. A través de los métodos, se pueden modificar los atributos de los objetos pertenecientes a determinada clase. Los métodos determinan el comportamiento de los objetos cuando reciben un mensaje, pero estos también permiten enviar mensajes a otros objetos, solicitando algún tipo de información o sugiriendo alguna acción.

Como una analogía a una clase podemos pensar en la lista de actividades que realizamos a diario



Esta lista la podemos organizar en grupos:



Cada grupo es una Clase con sus distintos métodos, los cuales pueden ser modificados sin alterar las otras actividades. Si por ejemplo, quisiéramos cambiar dentro del grupo "Ir al trabajo" el método o la manera en cómo nos desplazamos al trabajo (ej. taxi, bus, bicicleta, moto), lo podríamos hacer sin inconvenientes. De esta manera, los sistemas de software están estructurados, y a partir de este tipo de estructuras organizadas, es que se trabaja, de tal manera que, cada una de las partes que lo componen, sea cada vez más entendible.

Como se mencionó anteriormente, los objetos son los actores principales de un programa. Al momento de ejecutar un programa orientado a objetos, se crean los objetos que se requieren, se envían mensajes entre los objetos y la información es procesada de manera interna. Por último, cuando los objetos cumplen con su función, y ya no se requiere el uso de ellos, son eliminados, es decir, la memoria asignada a cada uno de ellos es liberada.

Como características de la programación orientada a objeto se tiene:

- ***Abstracción***: La abstracción es una característica que permite enfocarnos en la estructura externa de los objetos y no en sus detalles, y se puede definir como las características que identifican un objeto y lo diferencian de otros.

- ***Encapsulamiento***: El encapsulamiento es una característica que visualiza al objeto como una caja negra, sabemos que hay una información dentro del objeto, pero no sabemos cómo está organizada esa información.

- ***Herencia***: es una característica que permite compartir métodos y datos entre clases y subclases del objeto.

- ***Polimorfismo***: El polimorfismo es una característica que hace que se puedan hacer implementación de múltiples formas de un método dentro de una clase. Es decir, distintos métodos dentro de la clase a la cual se accede con un mismo nombre.

Todos los lenguajes que implementan parte o la totalidad de las características y los conceptos anteriormente descritos, son llamados **lenguajes orientados a objetos**. Este tipo de lenguajes se caracterizan porque permiten la definición de nuevos tipos de datos y de operaciones que se pueden realizar sobre estos. Algunos ejemplos de lenguajes orientados a objetos son: ADA, C++, Objective C, Java, Ruby, Python, Perl, PHP, C#, Kotlin, Visual Basic .NET, entre otros.

Los diagramas centrales del UML

Los diagramas son estructuras conformadas por elementos fundamentales, y es a partir de una combinación de estos diagramas, que se pueden formar los modelos que son vistos como una combinación de vistas dentro de nuestro sistema. Existen diferentes tipos de diagramas en un UML, los cuales son:

- Diagrama de clase
- Diagrama de objeto
- Diagrama de componente
- Diagrama de despliegue
- Diagrama de caso de uso
- Diagrama de estado
- Diagrama de actividad
- Diagrama de secuencia
- Diagrama de colaboración

Estos diagramas pertenecen a dos categorías, se pueden considerar, ya sea como **diagramas estructurales** o como **diagramas de comportamiento**.

Diagramas de comportamiento

Son diagramas que se enfocan en aspectos del sistema que contribuyen a satisfacer los requerimientos del mismo. Los diagramas de comportamiento son:

- ***Diagrama de caso de uso***: Permite modelar el núcleo de un sistema. Muestra una serie de actores del sistema y casos de uso.
- ***Diagrama de actividad***: Modela el flujo de una actividad.
- ***Diagrama de estado***: Ilustra el comportamiento interno relacionado con el estado de un objeto.
- ***Diagrama de secuencia***: Es un diagrama de interacción que describe el orden temporal de los mensajes que se envían entre objetos.
- ***Diagrama de colaboración***: Es un diagrama de interacción que describe el diseño organizativo de los objetos que reciben y envían mensajes.

Diagramas estructurales

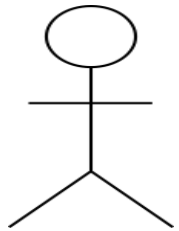
Este tipo de diagramas se centran en mostrar los aspectos estáticos de nuestro sistema. Dentro de esta categoría encontramos los siguientes diagramas:

- ***Diagrama de clase***: Permite ilustrar las clases y los paquetes, y la relación que hay entre ellos dentro del sistema.
- ***Diagrama de objeto***: Permite mostrar de manera estática una vista del sistema, mostrando la relación que existe entre objetos en ese estado.
- ***Diagrama de componente***: Permite mostrar de manera estática la relación que existe entre las componentes del *software* desplegable.
- ***Diagrama de despliegue***: Describe las componentes físicos del sistema.

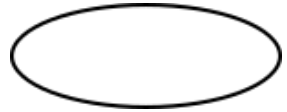
Elementos fundamentales

Algunos de los elementos fundamentales que se usan para la construcción de los diagramas y las vistas son:

- **Actor**: Este elemento representa un rol que juega el usuario final del sistema de *software*.



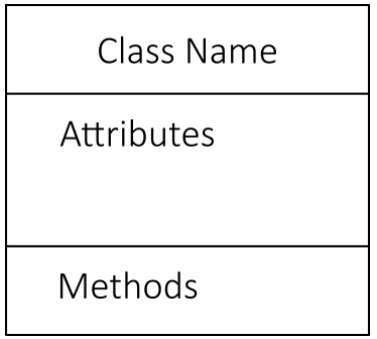
- **Caso de uso**: Elemento que representa las acciones que lleva a cabo el actor sobre el sistema de *software*.



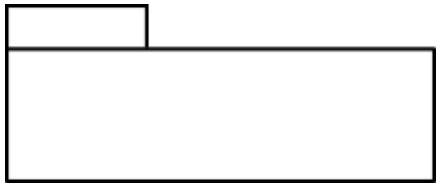
- **Colaboración:** Permite crear diagramas de clases que trabajan en conjunto unas con otras.

- **Objeto**: Es una instancia de una clase. Este elemento es un rectángulo en donde se puede especificar el nombre de la clase a la cual pertenece el objeto (subrayado y seguido de punto y coma), o se puede simplemente colocar el nombre del objeto (subrayado), o ambos (subrayado).

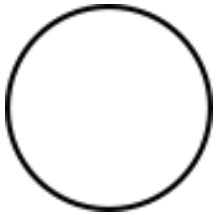
- **Clase**: Es una plantilla para la creación de objetos. Este elemento se compone de tres compartimientos, el primero representa el nombre de la clase, el segundo, los atributos de la clase, y el tercero, los métodos que contiene la clase. Los atributos y los métodos pueden ir adornados con un signo (+), un signo (-) o un signo (#), lo que indica que tales son públicos, privados o protegidos, respectivamente. Si el atributo o el método está subrayado, esto indica que es estático.



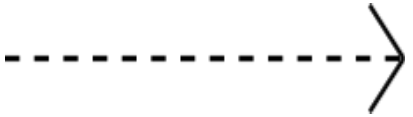
- **Paquete**: Este elemento puede contener cualquier otro tipo de elemento, se puede traducir directamente como un paquete en Java. Dentro de este elemento va el nombre que identifica el paquete.



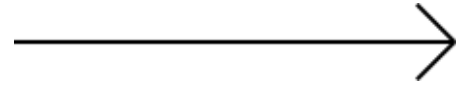
- **Interfaz**: Una interfaz es una lista o colección de acciones que posee una clase. Se traduce directamente a interfaces en Java. Este elemento se representa por un Ícono que es un círculo o por un diagrama de clase con el agregado <<interface>>.



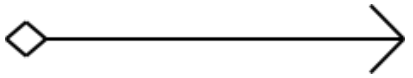
- **Dependencia**: Se usa para representar una relación de uso entre entidades (clases, objetos, paquetes, interfaz).



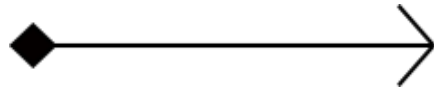
- **Asociación**: Representa la relación estructural entre entidades.



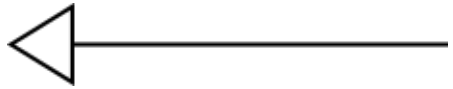
- **Agregación:** Es una forma de asociación que representa parte o toda la relación que hay entre dos clases.



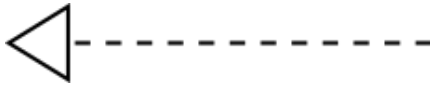
- **Composición**: Es una forma especial de agregación.



- **Generalización**: Permite ilustrar la relación existente entre un elemento más general y un elemento más específico. Esto permite modelar la herencia.



- **Realización**: Muestra la relación que especifica un contrato entre dos entidades. Una de las entidades define un contrato que es garantizado por la otra.



- **Anotaciones**: Permite incluir notas relacionadas con una explicación más amplia de los elementos y diagramas del UML.

Programas para trabajar con UML

Star UML

Es una herramienta para el modelamiento de software basado en los estándares UML (Unified Modeling Language) y MDA (Model Driven Arquitecture), que en un principio era un producto comercial y que hace cerca de un año pasó de ser un proyecto comercial (anteriormente llamado plastic) a uno de licencia abierta GNU/GPL.

El software heredó todas las características de la versión comercial y poco a poco ha ido mejorando sus características.

Link de referencia bibliográfica: Publicado el julio 1, 2011 por Sergio Zamenfeld en Software https://www.brainlabs.com.ar/novedad/staruml-una-herramienta-para-modelado/

link de descarga: https://staruml.io/

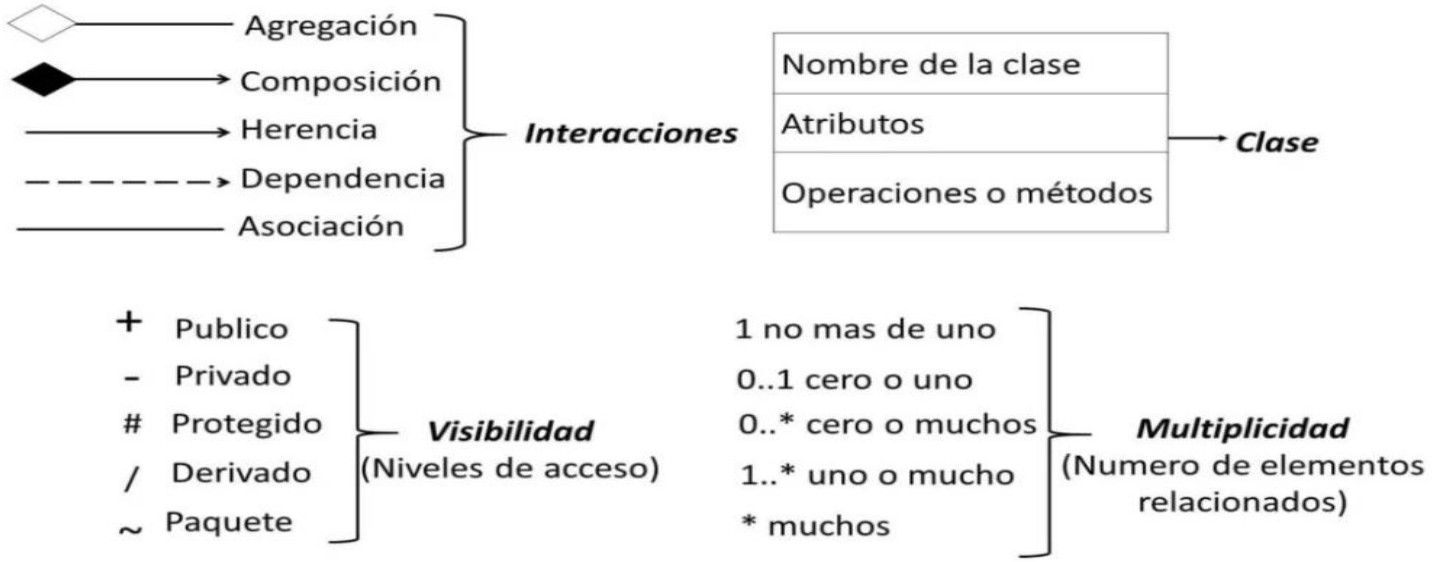
ArgoUML

“ArgoUML” es una herramienta desarrollada en Java que permite crear modelos UML compatibles con los estándares de la versión 1.4 de este lenguaje.

Los tipos de diagrama que se pueden crear con ArgoUML son nueve: diagrama de clases, diagrama de estados, diagrama de actividad, diagrama de casos de uso, diagrama de colaboración, diagrama de despliegue y diagrama de secuencia.”

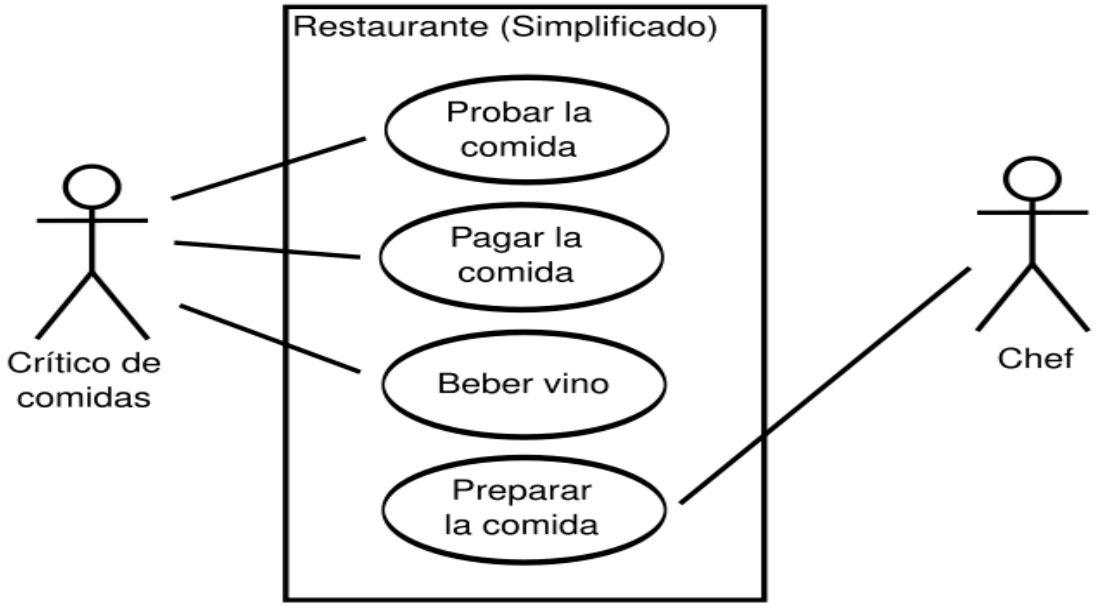
Link de referencia y descarga: https://argouml.uptodown.com/windows

Elementos y símbolos en los diagramas de clases UML

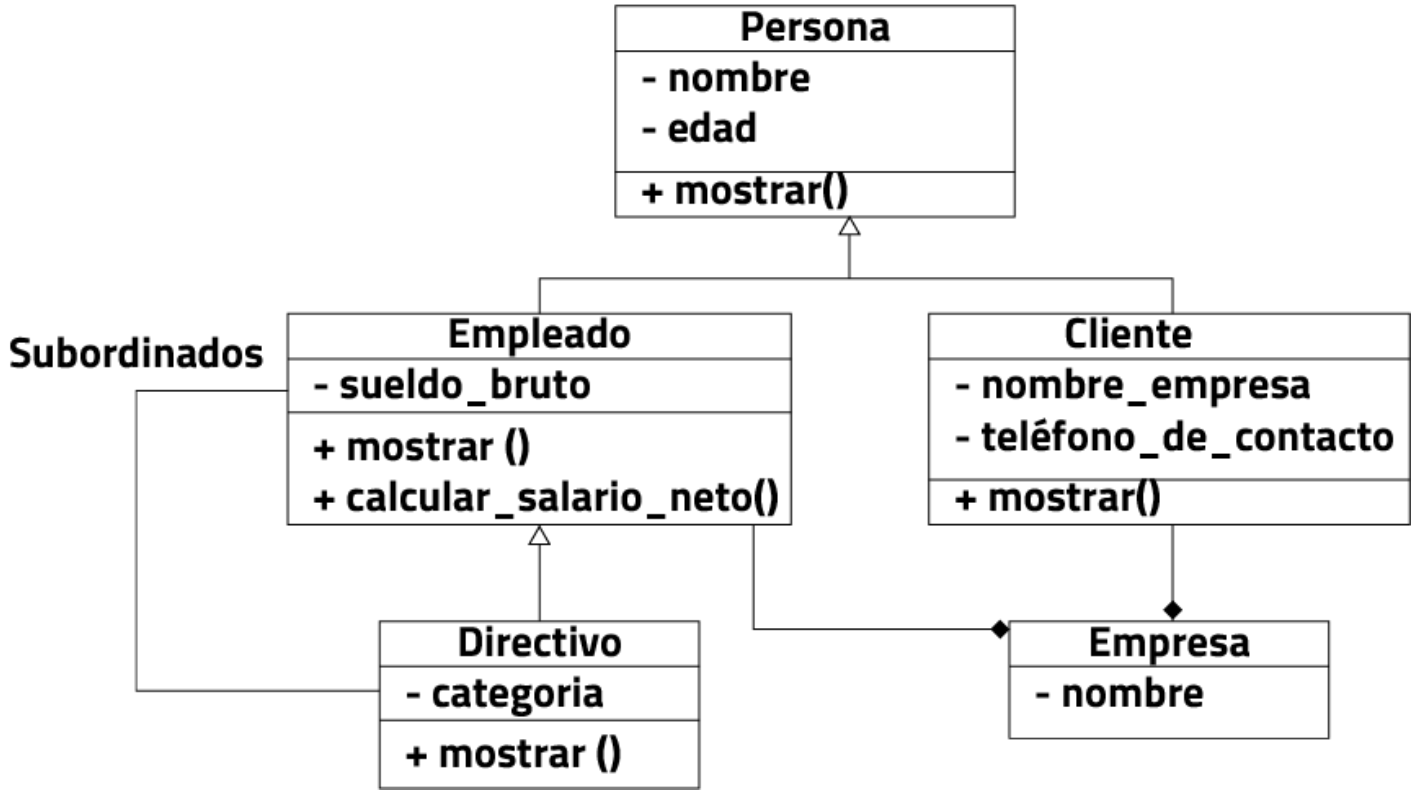


Link ejemplo caso de uso: https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso

Ejemplo casos de uso:



Ejemplo diagrama de clases: https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html



Objetos y clases

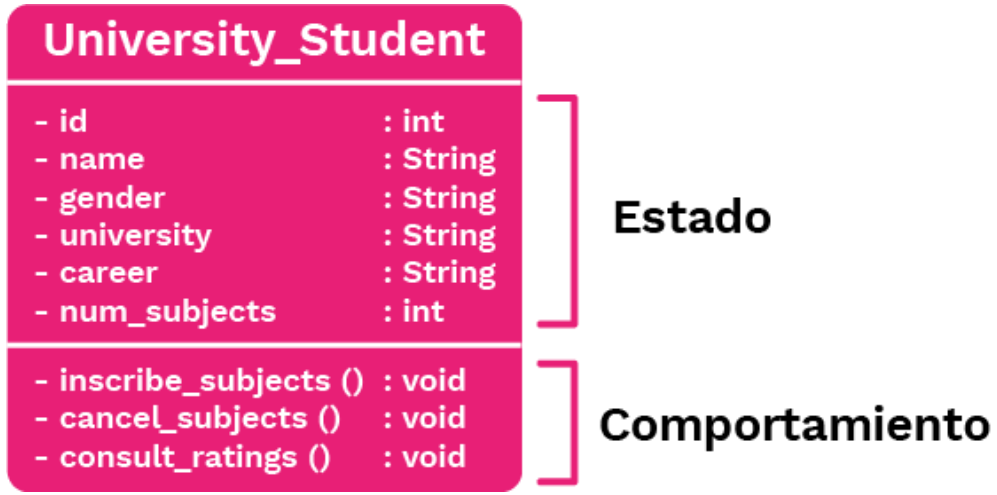
Link bibliografia: https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html Creado y modificado por: Laura Álvarez, Helmer Avendaño, Yeison García, Sebastián Morales,

Edwin Bohórquez, Santiago Hernández, Sebastián Moreno, Cristian Orjuela

Clase

La clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede mencionar que una clase es una plantilla genérica para un conjunto de objetos de similares características. Una clase define el estado y el comportamiento que todos los objetos creados a partir de esa clase tendrán.

Ejemplo:



Ejemplo desarrollo de Java:

```java
public class UniversityStudent {
    int id;
    String name;
    String gender;
    String university;
    String career;
    int numSubjects;
    public UniversityStudent(int id, String name, String gender,
        String university, String  career, int numSubjects) {
        this.id = id;
        this.name = name;
        this.gender = gender;
        this.university = university;
        this.career = career;
        this.numSubjects = numSubjects;
    }
    void inscribeSubjects() {
      // TODO: implement
    }
    void cancelSubjects() {
       // TODO: implement
    }
    void consultRatings() {
       // TODO: implement
    }
}
```

Objetos

Los objetos de software, al igual que los objetos del mundo real, también tienen características y comportamientos. Un objeto de software mantiene sus características en uno o más atributos e implementa su comportamiento con métodos.

Es una entidad real o abstracta, con un papel definido en el dominio del problema.

Un **objeto** es una instancia de una clase que tiene: identidad, estado(atributos) y comportamiento(métodos).

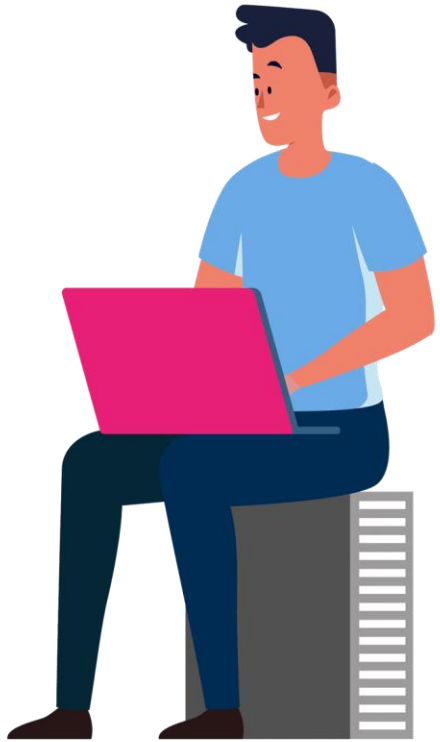
Para el ejemplo anterior de la Clase UniversityStudent (Estudiante universitario), la creación de objetos se haría de la siguiente forma:

Ejemplo en Java

```
UniversityStudent  student = new UniversityStudent (123, "Pepe", "masculino", | "Un", "Medicina", 8);
```

Los niveles de visibilidad (modificadores de acceso)

Las clases, los métodos y los atributos, pueden ser declarados de acuerdo a cierto nivel de visibilidad. Existen distintos niveles de visibilidad dentro del lenguaje de programación Java, que restringen el acceso dentro de un sistema de *software* tanto a las clases, como a los atributos y a los métodos. Estos niveles de acceso pueden ser modificados mediante un modificador que acompaña a la declaración de cada uno de estos elementos. Cada modificador ofrece un nivel de visibilidad dentro del sistema de software que se esté desarrollando, cuyas características se muestran en la siguiente tabla.



Tab. 1: Modificadores de acceso dentro del lenguaje de programación Java.

Si deseas conocer un poco más acerca de los modificadores de acceso puedes consultar el siguiente enlace: https://www.programarya.com/Cursos/Java/Modificadores-de-Acceso

UML	Modificador	Clase	Paquete	Subclase	Todo el programa
+	**public**	Si	Si	Si	Si
#	**protected**	Si	Si	Si	No
~	**Sin modificador**	Si	Si	No	No
-	**privado**	Si	No	No	No

Revisión de lo aprendido

1. El lenguaje de programación C, sirve para la programación orientada a objetos

    - Falso
    - Verdadero

Revisión de lo aprendido

2. Un objeto hace parte de una clase

    - Verdadero
    - Falso

Revisión de lo aprendido

3. El constructor debe tener un nombre igual al nombre de la clase

    - Verdadero
    - Falso

Revisión de lo aprendido

4 . El Modificador de acceso protected:

a. No es de acceso público, solo se puede acceder desde la clase y sus subclases
b. Solo se pueden acceder a métodos o atributos si están en la misma clase
c. Se puede acceder a los métodos y atributos sin ninguna restricción

Revisión de lo aprendido

5. Con el Modificador de acceso private:

   A. Solo se pueden acceder a métodos o atributos si están en la misma clase
   B. No es de acceso público, solo se puede acceder desde la clase y sus subclases
   C. Se puede acceder a los métodos y atributos sin ninguna restricción

Revisión de lo aprendido

6. Un objeto es una instancia de la clase

    - Verdadero
    - Falso

Revisión de lo aprendido

7. El constructor puede devolver un void

- Falso
- Verdadero

Revisión de lo aprendido

8 . La clase es una plantilla genérica para un conjunto de objetos

- Verdadero
- Falso

Revisión de lo aprendido

9. El Modificador de acceso public:

   A. Puede acceder a los métodos y atributos sin ninguna restricción
   B. No es de acceso público, solo se puede acceder desde la clase y sus subclases.
   C. Solamente se puede acceder a métodos o atributos si están en la misma clase.

Revisión de lo aprendido

10. El Diagrama de despliegue es elemento que representa las acciones que lleva a cabo el actor

- Falso
- Verdadero

Revisión de lo aprendido

11. El diagrama de componente: describe las componentes físicas del sistema.

- Falso
- Verdadero

Revisión de lo aprendido

12. El paquete es una plantilla para crear objetos

- Falso
- Verdadero

El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

CICLO 2

EJE TEMÁTICO 2

PROGRAMACIÓN ORIENTADA A OBJETOS Y UML

Universidad Industrial de Santander

MisionTIC 2022