**INTRODUCCION AL PARADIGMA DE OBJETOS.**

**MODULO I**

**Paradigma**:

Teoria o conjuntos de teorias que sirven de modelo a seguir para resolver problemas.

La POO define los programas por medio de “**clases de objetos**”

**Los objetos combinan:**

- Identidad (nombre de la clase)

- Estado (caracteristicas)

- Comportamiento ( acciones )

Los objetos conviven en un programa y se comunican a traves de mensajes.

Ej de lenguajes de POO

- Java

- PHP

- Phyton

- C#

**Ventajas de la PPO**

- Fomenta la reutilizacion del codigo

- Crear sistemas complejos

- Semejanza al mundo real

- Agiliza el desarrollo de sofware

- Facilita el trabajo en equipo.

**Conceptos basicos de la PPO**

**Objeto**: Entidad provista de un conjunto de atributos (datos) y comportamiento (metodos)

**Clase:** Es el modelo en que se definen la propiedades y comportamiento de un tipo de objeto. (molde) Se usa para ir creando objetos. Esta compuesta por atributos (datos), tambien llamadas variables de instancia o variables miembro que nos indican en que estado se encuentra el objeto. Tambien se compone de metodos (funcones o procedimientos) que indican cual es el comportamiento.

Los atributos de la clase son privados al objeto y su acceso se realiza mediante los metodos definidos para manejar esos atributos.

**Metodo (operacion):** Son acciones contenidas por el objeto, que se asignan en la clase. Determina el comportamiento que puede tener un objeto, el cual fue referido en la clase. Es un algoritmo asociado a un objeto o una clase de objetos que se ejecuta tras la recepcion de un mensaje. Son generalmente verbos. Pueden admitir variables como parametros de entrada y retornar valores de salida o retorno. Los metodos u operaciones modifican el estado de los objetos.

Valores de **entrada** (**datos**): Son valores enviados a un metodo para realizar la operación.

Valores de **salida** o retorno (**informacion**): Es el valor devuelto por un operación luego de realizar un procesamiento. Todos los valores de salida deben tener un tipo de dato asociado. Solo devuelven 1 solo objeto.

**Mensaje:** Es la comunicación hacia un objeto para que este ejecute uno de sus metodos para obtener un resultado.

**Propiedad o atributo:** Es un tipo de dato que define a nuestro objeto. Son las caracteristicas y normalmente cuentan con un dato asociado. Siempre seran sustantivos que haran referencia a nombres, tamaños formas y estados. Las clases definen los atributos y los objetos los completan !

**Estado interno:** Segun sus valores en un momento dado representan el estado de ese objeto en ese momento. Ese estado puede ser modificado a traves de un metodo definido en la misma clase a cual pertenece el objeto.

**Componentes de un objeto:**

- Identidad

- Atributos

- Metodos

- Relaciones

**Representacion de un objeto:**

Va a ser representado por medio de una entidad compuesta por sus atributos y funciones.

**Comunicacion entre objetos.**

Emisor le pide algo al receptor y puede que obtenga una respuesta o no.

**Caracteristicas de la PPO**

- **Abstraccion** (nos permite seleccionar ciertas caracteristicas del objeto atributos o comportamientos según necesitemos)

- **Encapsulamiento** Nos permite reunir todos los elementos que pertenecen a una entidad particular.

- **Herencia** Las clases se relacionan entre si y forman una jerarquia de clasificacion. Los objetos heredan la propiedades y comportamiento de todas las clases a las que pertenecen.

- **Polimorfismo** Se pueden definir clases diferentes en la misma jerarquia de herencias que tienen metodos denominados de forma identica pero se comportan de manera distinta.

**MODELOS**

Los Modelados previos de los sistemas de informacion sirven de intermedio para poder entender el problema y la solucion en la orientacion a objetos

**Diagrama de clases (el mas importante)**

1- Identificar clases y objetos del dominio del problema.

2- Definir el comportamiento de los objetos.

3- Asociar estaticamente los objetos.

4- Definir la jerarquia de herencia de clases

**DIAGRAMA UML** es una herramienta para diagramas de modelado que incluye casos de usos, actividades, iteracion, estados e implementacion. (*drawio*: sofware libre para diagramar)

**MODULO II**

**RELACOINES ESTRUCTURALES**

Son relaciones que se dan entre los objetos de diferentes clases.

**Asociacion** Es una relacion estructural que especifica que los objetos de un elemento estan conectados con los objetos de otro elemento. Dada una asociacion entre dos clases se pude navegar de un objeto de una a un objeto de otra.

**Multiplicidad** de una asociacion determina cuantos objetos de cada tipo intervienen en la relacion. Es el numero de instancias de una clase que se relaciona con una instancia de otra clase. Cada asociacion tiene **dos multiplicadores** (uno en cada extremo).

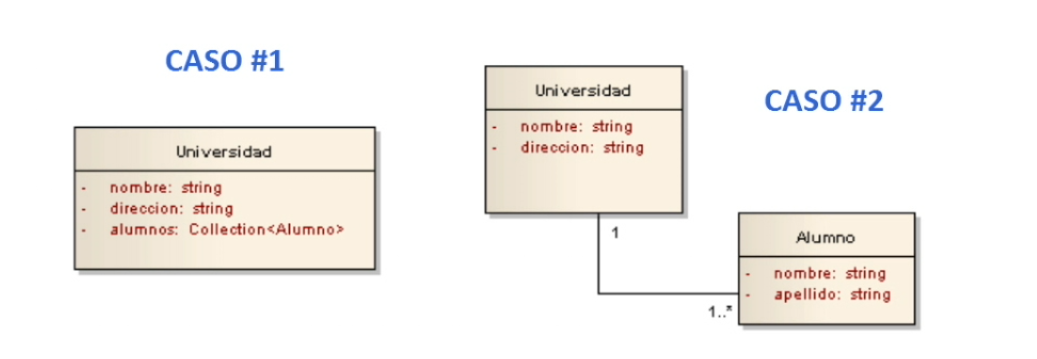
Ej 1 ------> 1,N

De las asociaciones surgen 2 tipos de relaciones:

**Relacion Simple** 1 – 1 (uno a uno)

**Relaciones multiples** 1 – N (uno a muchos)

En las relaciones multiples, se puede representar una clase con una colleccion de otra clase adentro, o por otro lado una clase con una relacion multiple a otro objeto separado.



**SOBRECARGA DE OPERACIONES**

Es la aparicion de metodos dentro de una misma clase que se llaman igual, pero realizan acciones diferentes.

Nos permite utilizar el mismo nombre para dos operaciones distintas, que aceptan distinta cantidad y/o tipos de parametros y tienen distinta funcionalidad. El tipo de devolucion no puede ser el mismo. El compilador distingue estas operaciones según el tipo y cantidad de parametros.

(PHP, Python y JS no implementan sobrecarga de metodos)

Ej: class automovis {

void acelerar () { velocidad += 10; }

void acelerar (int km) { velocidad += km; }

void acelerar (int km, boolean nitro); {

if (nitro) { acelerar(km\*2); }

else { acelerar(km); }

}

**VISIBILIDAD**

Determina el alcance (scope) que tiene un atributo u operacion. Es desde donde puedo acceder a una clase, atributo u operación.

El alcance es fundamental en la POO, ya que permite definir el encapsulamiento de la informacion y el comportamiento. Los metodos, atributos y clases deben tener un solo alcance, tambien llamado **tipo de acceso. (**en UML,**+ publico, - privado)**

**Modificadores de visibilidad**

**- defaul (nivel paquete):** Acceso por defecto. Si un elemento clase, metodo o atributo tiene acceso por defecto, solo las clases de su mismo paquete tendran acceso a ellos.

**- public:** Si un atributo, metodo o clase tiene visivilidad publica, es accedido desde cualquier clase. Tener en cuenta que solo podremos ver lo que esta marcado como publico, si una clase es publica y sus atributos no, solo podremos ver la clase sola. Habria que marcar como publicos los atributos que querramos.

**- private:** Solo es aplicable a metodos y atributos. Solo pordra ser accedido por los metodos definidos en la misma clase. Cuando declaramos todos los atributos de un objeto como privado lo estamos **encapsulando.** Para acceder a atributos privados podemos definir las **operaciones** internas del objeto como **publicas** para acceder entonces a los atributos.

**- protected:** Se utiliza en la herencia y es aplicable a atributos y a operaciones. Un atributo o metodo protegido puede ser accedido por cualquier otra clase de su mismo paquete y **ademas** por cualquier **sub clase** de ella independientemente del paquete en que se ecuentre. Las subclases pueden acceder a los miembros protegidos a traves de la **herencia**. Basicamente los metodos protected de una clase padre son accedidos por metodos de una clase hija desde cualquier paquete.

**ENCAPSULAMIENTO**

Es el ocultamiento del estado de un objeto. El estado de un objeto es el valor de sus atribtutos. Permite que todo lo referente a un objeto quede aislado dentro de este y solo se pueda acceder a ello a traves de sus metodos.

Los metodos para exponer el objeto al exterior llevan modificador publico y los atributos privado.

Beneficios:

- **Proteccion de datos sensibles**. Los atributos del obj no pueden ser modificados por otras clases u operaciones externas.

- **Facilidad y flexibilidad en el mantenimiento** de apps. Solo tenemos que modificar los metodos, asi no aleramos a otros objetos relacionados.

**GETTERS Y SETTERS** (Minutuo 19 clase 3)

Los metodos getter y setter, son metodos de acceso publico a atributos privados.

- **setter:** Es un “metodo” utilizado para **setear** el valor de un atributo. No retorna nada pero ingresa un parametro de igual tipo al dato del tributo que trata. Lo que hace es asignar el valor del parametro al atributo correspondiente.

- **getter:** Es el metodo usado para **obtener** el valor de un atributo. No posee atributo alguno pero **retorna** un dato. Lo que hace es resolver el valor del atributo correspondiente.

Ej: class Banco {

private String Nombre;

public void setNombre(string n) {

nombre = n;

}

public string getNombre() {

return nombre;

}

En la misma clase se define un get y un set por cada atributo con igual tipo de dato. El set recibe un parametro y el get devuleve un valor. Basicamente, con los setters accedemos al atributo del objeto enviando un parametro, para modificarlo/inicializarlo. Y con el getter los usamos en los metodos para acceder al valor del atributo establecido con el setter.

En UML tenemos,

Clase

atributos

metodos

setter

getter

**CONSTRUCTORES**

- Toda clase tiene un constructor

- Los objetos son construidos a partir de una clase (molde). Todo objeto depende de una clase. **Son instancias de una clase** y tienen un estado.

- Para construir un objeto se necesita un constructor. El constructor crea un objeto en memoria y se le asigna a la variable con el nombre que elijamos.

- Un constructor es el metodo invocado en el momento en que se crean las instancias de una clase, o sea los objetos.

- **Tiene el mismo nombre que la clase** y para invocarlo se utiliza la palabra reservada **new.**

**- No puede tener ningun tipo de devolucion.** Ni siquiera void.

**-** Se pueden sobrecargar, o sea, una clase puede terner mas de un constructor y por lo tanto distintas formas de inicializar los atributos. Distinta cantidad de parametros para que sean validos, o distinto tipo de dato o distinto orden.

- Toda clase debe tener al menos 1 constructor. Si no lo asignamos el compilador lo establecera uno vacio por defecto.

- En sobrecarga de constructores, podemos tener un constructor vacio para ir asignandole valore a los atributos a la vez que los vamos cargando, y a la par podemos tener uno o varios constructores sobrecargados con parametros ya asignados previamente (siempre y cuando no tengasmos problemas con las firmas de parametros). Por ejemplo, no podemos tener 2 constructores y que cada uno reciba 1 string, porque el programa no va a saber diferenciar. La sobrecarga dede tener constructores diferenciados por tipo de dato y/o cantidad de parametros, y/O orden de los mismos.

- Los constructores parametrizados evitan el uso de inicializacion con setters y getters. Y nos da la seguridad de tener que ingresar todos los parametros que requiere o arrojara error.

Ej: ClassName objectName = new Constructor()

el constructor es el nombre de la clase

**Clase programa**

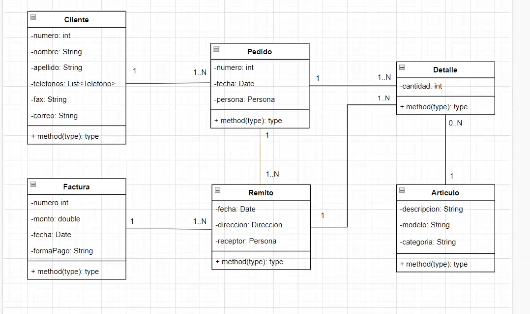
La clase programa es la que instancia el primer objeto a traves de un metodo llamado main(), que se invoca automaticamente al iniciar nuestra aplicación.

**DESTRUCTORES**

Se utilizan para destruir una instancia de una clase y liberar memoria.

Su utilizacion depende del lenguaje de programacion. En Java por ejemplo no hay destructores, la liberacion de memoria es llevada a cabo por el **garbage collector** cuando las instancias de los objetos quedan desreferenciadas. Ejemplo, a un obejto creado le asignamos el valor null, entonces queda sin referencia.

EJERCICIO LABORATORIO ALUMNI – Relaciones, atributos y clases.



**MODULO III – (clase 4, minuto 00:22)**

**HERENCIA**

Es la capacidad de crear clases (subclase, clase derivada o hija) que adquieren automaticamente atributos y metodos de clases ya existentes ( clases padre, superclase, clase base ) pudiendo al mismo tiempo añadir metodos y atributos propios. Las clases hijas pueden ser a su vez clases padres de otras clases, heredando estas ultimas todas las propiedades y metodos por cascada de todas las clases superiores.

La Herencia nos permite la reutilizacion de codigo. Es el mecanismo mediante el cual una clase hereda todo el comportamiento y atributos de otra clase.

**HERENCIA SIMPLE**

Una clase hereda las caracteristicas de **una sola** clase padre.

**HERENCIA MULTIPLE**

Una clase hereda metodos y atributos de dos o mas clases. Esto va a depender del lenguaje de programacion, Java por ej. no permite la herencia multiple.

En la herencia entre dos clases se establece una relacion “**es un”**, o sea que un objeto de una subclase es tambien un objeto de la superclase.

Lo **atributos privados** de una clase seguiran siendo privados en la subclase, y solo se podra acceder a ellos desde los metodos publicos. Para **acceder** a los **metodos privados de una superclase** desde una clase hija, podemos hacerlo a traves de los metodos publicos **get y set** si esta los tuviese.

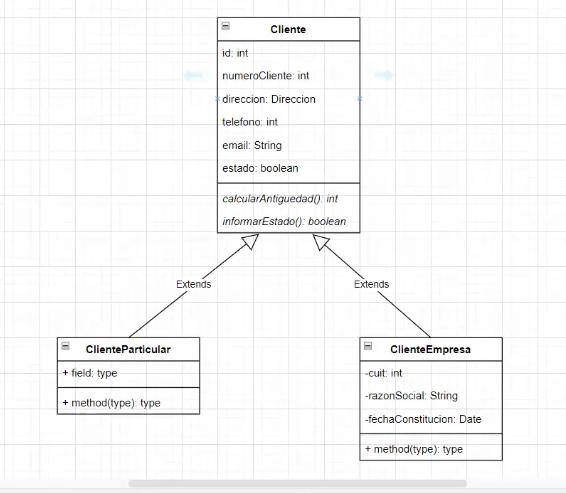
Ejemplos de declaracion de herencia

**Java** class Estudiante extends Persona

**PHP** class Estudiante extends Persona

**C#** class Estudiante : Persona

**Python** class Esttudiante(Persona)



**EJECUCION DE CONSTRUCTORES.**

Nota: cada vez que se crea un objeto de una clase, **primero** se ejecuta el constructor de la **superclase** y por **ultimo** el de la **subclase**. El compilador de java x ejemplo invoca al constructor de la clase padre con la palabre reservada **super()**, que a su vez es añadida automaticamente en cada constructor como primera linea del mismo

public class Empleado extends Persona {

private string atributoA;

public Empleado() {

super();

} (constructor vacio)

public Empleado(param1, param2, param3) {

super(param1, param2);

this.atributoA = param3

}

(constructor parametrizado con super de subclase)

\* primero invocamos al constructor de la clase padre y luego a la inicializacion de los propios atributos. El constructor vacio la al constructor vacio de la clase padre inicial (clase object).

OTRO EJEMPLO DE LLAMADA DE CONSTRUCTOR

public class Empleado extends Persona {

private string atributoA;

public Empleado() {

super();

} (constructor vacio)

public Empleado(param1, param2) {

super(param1, param2);

}

public Empleado(param1, param2, param3) {

this(param1, param2);

this.atributoA = param3

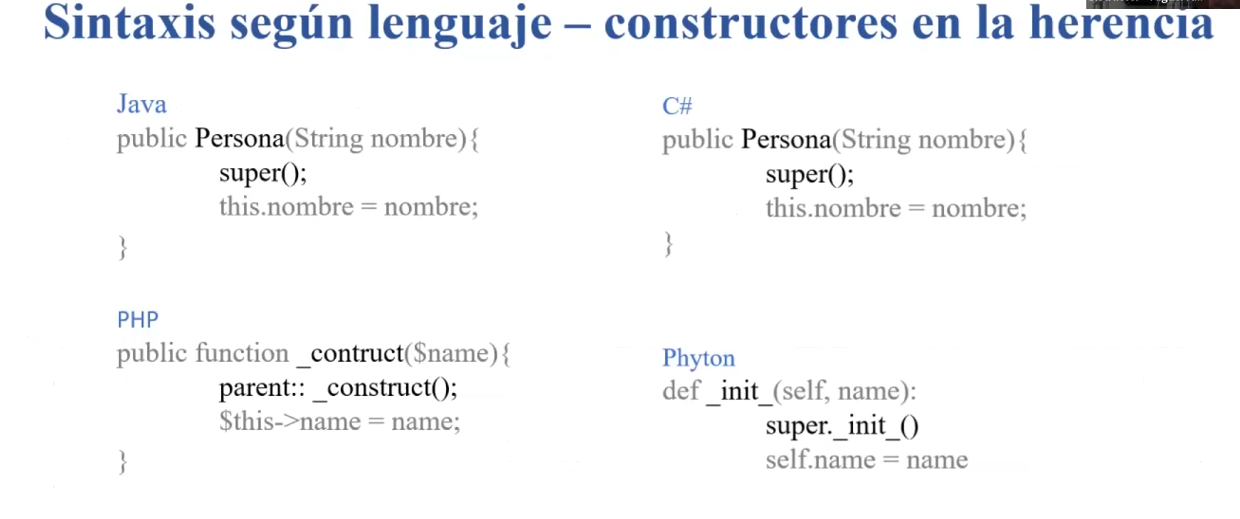
Otra opcion por ejemplo si ingresamos un nuevo atributo es, en vez de llamar al constructor de la superclase y luego al atributo this.atributoxxx, es agregar otro constructor con un parametro mas, y con this llamar en este caso a constructor anterior dentro de la misma clase y ahi agregar despues el this.atributoxxx.

**MODIFICADORES DE ACCESO EN LA HERENCIA.**

- Si los datos de la clase base son **privados**, tambien lo seran en la clase derivada y su acceso sera mediante metodos publicos.

- Si los datos en la clase base son **publicos** tambien los seran en la clase derivada y su acceso sera directo sin utilizar metodos. Se accede a traves del nombre.

- Si los datos en la clase base son **protegidos**, en la clase derivada seran **publicos**.



**CLASES FINALES**

Son las clases que no queremos que sean heredadas o extendidas por otra clase.

Con el modificador **final** delante de **class**, evatamos que sea heredada o extendida por otra. Esto se usa cuando queremos obetenr una clase con atributos ya definidos y que no se vuelvan a modificar en un futuro.

Lo **mismo** **aplica** para los **metodos** que son heredados por otras clases. Un metodo con modificador “final” **no se puede sobreescribir** desde una subclase.

**SOBRE ESCRITURA DE METODOS.**

Para poder sobreescribir un metodo heredado o publico, hay que tener en cuenta ciertos parametros:

- Misma firma ( mismo nombre e iguales parametros )

Por ejemplo si quiero cambiar la accion que devuelve un metodo que es heredado de un superclase, pero en una de las subclases realiza una accion diferente a la original, podemos cambiar el codigo de ejecucion pero la firma del metodo debe ser la misma.

Ejemplo de extraer en caja de ahorro y en cta corriente, ya que la ultima tiene en cuenta el descubierto.

Si cambiamos la firma del metodo, entonces ya pasariamos a tener una sobrecarga de metodos y no una sobreescritura.

El metodo sobreescrito puede tener un **modificador** **nenos** **restrictiva** que el de la clase padre pero **nunca** una **mayor**.

El protected – public.

**ATRIBUTOS DE INSTANCIA Y DE CLASE.**

**Atributos de instancia**: Son los que definimos en una clase. Cada vez que creamos un objeto a partir de una clase, en ese objeto se crea una copia de esos atributos. Estos se invocan a traves de una variable que mantiene una referencia a un objeto creado en memoria.

**Atributos de clases:** Es un atributo **estatico** que es compartido por objetos de diferente clase que lo crean. O sea que si varias clases crean objetos que comparten un mismo atributo, no se realiza una copia de ese atributo por cada instanciacion sino que cada objeto comparte un valor unico de ese atributo. Puede ser modificado accediendo de cualquier objeto que contenga ese atributo.

Para definir un atributo estatico o de clase, anteponemos el modificador **static** al nombre del atributo. A diferencia de un campo standard que lleva el modificador protected, estos campos elasticos suelen llevar un modificador que permita el acceso directo, ej **public, protected o ninguno.**

Esto resulta util cuando tenemos un valor de atributo que es igual para varios objetos que lo poseen.

**Metodos de instancia:** Son metodos que pertenecen a un objeto en particular e impactan solamente en el comportamiento de este objeto.

**Metodos de clase:** Son metodos que pertenecen a la clase y no a un objeto de instancia de clase. Son metodos compartidos por todos los objetos. Tambien llamados metodos estaticos. Se definen con la palabra **static.**

**Consideraciones:**

- Los metodos estaticos **no pueden acceder** a atributos que **no sean estaticos**.

- No podemos dentro de un metodo estatico hacer uso de **super** y **this.**

**CLASES CONCRETAS Y CLASES ABSTRACTAS.**

**Clases Concertas:** Es una clase que se puede instanciar.

**Clase abstracta:** Son clases que tienen algun metodo abstracto. Los metodos abstractos estan declarados pero no definidos. Se especifica nombre, parametros y tipo de devolucion pero **no incluyen codigo**. Estos metodos abstractos se incluyen cuando su funcion no esta determinada y su comportamiento luego sera definido por las subclases de la clase abstracta mediante la sobreescritura del metodo. Se definen con la palabra reservada **abstract.**

**No se pueden crear objetos** de una clase abstracta, pero si tiene constructores porque estos los necesitan invocar las clases hijas para crear objetos.

Las clases abstractas suelen crearse para definir el formato de una clase, para que luego sean transformadas por las subclases implicadas.

Una subclase esta obligada a sobreescribir todos los metodos abstractos heredados, caso contratio de debera declarar a la subclase tambien como abstracta.

**INTERFACES**

Es un conjunto de **metodos abstractos** y **constantes publicas.** La finalidad de una interfaz es la de definir el formato que deben tener determinados metodos que van a implementar ciertas clases. Solo se establecen nombre, parametros y tipo de devolucion, pero no se establece el funcionamiento.

Se definen con la palabra reservada **interface.**

Caracteristicas:

- Todos los metodos son publicos y abstractos.

- Se pueden definir constantes, que seran publicas, estaticas y finales.

- Un interfaz no es una clase.

- No tiene constructores

Cuando una clase desea implementar la interfaz, en vez de extend, se usa la palabra reservada **impements.**

Public class MiClase **implements** MiInterface { }

Al igual que cuando se heredan clases abstractas, cuando una clase **implementa** una **interfaz**, esta **obligada** a **definir** (implementar) el codigo de **todos** los **metodos** **existentes** en la misma. De no ser asi, la clase se declarara como abstracta.

Una clase puede implementar mas de una interfaz, siempre teniendo que implementar los metodos de todas la interfaces, a la vez que heredar de otra clase.

Public class MiClase extens Superclase **implements** Interfaz1, Interfacz2, { }

Una **interfaz** tambien puede **heradar** de **otra** **interfaz**. La intefaz que extiende de otra hereda todos sus metodos abstractos.

Public interface MiInterface **extends** Interface1, Interface2 {}

**INTERFACES AVANZADAS EN JAVA**

- Java 8, permite interfaces con metodos default

- Java 9, permite interfaces con metodos private

**Diferencias entre clases e interfaces abstractas.**

**Clase abstracta: (nos referimos a misma familia de objetos)**

- Al menos 1 metodo abstracto.

- Atributos

- Metodos con comportamientos

- Permite definir nuevas clases pertenecientes a la misma familia de objetos.

**Interfaces abstractas: (referidas a diferente familias de obj)**

- Metodos abstractos

- Constantes

- Nuevos modificadores (default y private) permiten definir metodos con comportamiento.

- Los metodos se pueden implementar en muchas familias de objetos.

**POLIMORFISMO**

Es la posibilidad de que una clase presente un comportamiento distinto deacuerdo a una situacion.

**Polimorfismo con redefinicion:** Una clase modifica metodos heredados de una superclase porque requieren otro comportamiento. (redefinicion de metodos)

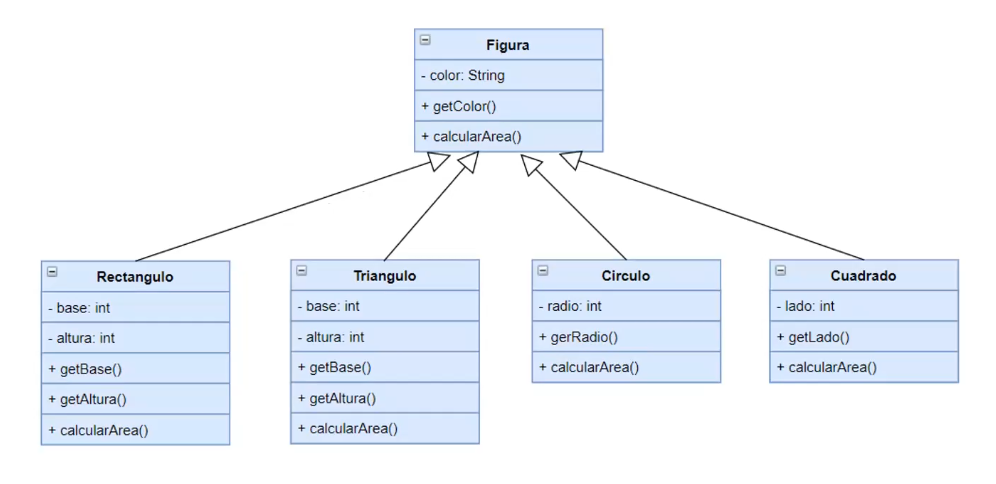
**Polimorfismo sin redefinicion:** Una clase posee varios metodos llamados iguales pero con diferentes firmas (sobrecarga de operaciones)

**Asignacion de objetos a variables de SuperClase**

Se puede asignar un objeto de una clase hija a una variable de su Superclase. Entonces se puede utilizar esta variable para invocar los metodos del objeto que esten definidos o declarados en la superclase, por no los que solo existan en la clase a la que pertence el objeto.

Esto propicia la posibilidad de que una misma instrucción llame a distintos metodos dependiendo del objeto almacenado en la variable de la clase padre.

*“Esta es la ventaja del polimorfismo, usar una misma expresion para invocar diferentes versiones de un mismo metodo, determinando en tiempo de ejecucion la version del metodo a ejecutar”*

La principal ventaja de polimorfismo es la reutilizacion de codigo.

Ejemplo,

- la clase figura es abstracta porque tiene un atributo

- clacularArea() dependera de cada clase hija.

- Asignamos a una variable del tipo superclase el objeto de la clase hija.

Entonces, según el tiempo de ejecucion, una misma variable (ej: **Figura**) de la clase padre, ejecutara la funcion que le corresponda según el objeto de la clase hija que se le asigne, cuya funcion estara determinada en la clase hija en cuestion.

Por ejemplo, asignamos el objeto circulo en la variable Figura y entonces cuando ejecutemos figura ejecutara el metodo referente al circulo. Luego asignamos otro objeto a la variable Figura y ejecutara el metodo correspondiente al nuevo objeto.