Interfaces

Una interfaz es un medio común para relacionar clases que puedan no estar naturalmente relacionadas.

...¡¿Qué?!

Interfaces

Las interfaces son como un **contrato**.

Establecen qué **métodos** deberán implementar todos los que adhieran a este contrato.

De esta forma, si conozco el contrato, y se que cierta clase “respeta” ese contrato, entonces sabré sin lugar a dudas que a esa clase puedo invocarle los métodos que describe el contrato.

Interfaces

En este **contrato**, la interfaz **DEFINE:**

- ✅ los nombres de los métodos
- ✅ su tipo de retorno
- ✅ y tipo y cantidad de parámetros que reciben.

En este **contrato**, la interfaz **NO DEFINE**:

- ☒ implementaciones (en ningún método).
- ☒ atributos

Interfaces

Las Interfaces no se podrán instanciar.

Al igual que las clases abstractas, no podrá haber objetos del tipo de la Interfaz.

Interfaces

Recordamos:

“Una interfaz es un medio común para relacionar clases que puedan no estar naturalmente relacionadas.”

Pensémoslo otra vez...
Si quiero relacionar las clases **Perro**, **Gato**, **Canario…** claramente las puedo hacer heredar de una Clase Abstracta **AnimalDomestico**.
Son clases **relacionadas**, y es natural una abstracción/generalización para relacionarlas.

Interfaces

Pero si quiero relacionar las clases **Perro, Caballo, Ferrari, Maratonista, ForestGump...**

Estas clases **no están naturalmente relacionadas.** No puedo pensar en una herencia que las relacione.

Pero supongamos el escenario de un **VideojuegoDeCarreras**, donde me interesa que todas las clases puedan competir.
Entonces, puedo pensar que todos tengan el método correr();

Lo podemos solucionar elegantemente con la **Interfaz Corredor.**

Interfaces

Así, en la **Interfaz Corredor**, definiremos el método **correr()**;

- El método correr() no tendrá implementación (no tendrá código).
- La interfaz no tendrá atributos.
- No se podrá instanciar esta interfaz.

Ahora podremos relacionar **Perro, Caballo, Ferrari, Maratonista, ForestGump**, con la **interfaz Corredor**.

Y todas las clases serán también del tipo **Corredor**.

Interfaces

Y como ahora todas las clases son también del tipo **Corredor**.

En la clase **VideojuegoDeCarreras**, podré tener una **lista o array con elementos del tipo Corredor**.

Y a todos estos elementos, con confianza, les podré pedir elemento1->correr(), elemento2->correr()...

Interfaces

A la clase **VideojuegoDeCarreras** no le importa si es un Perro, un Caballo o ForestGump. Como son todos de **tipo Corredor**, a todos podrá decirles correr();

Todos respetan el **contrato** establecido por la **interfaz Corredor.**

Interfaces

Otro ejemplo:
Tengo que diseñar el sistema de una *clínica de terapias alternativas anti-stress*.

La clínica tiene un método innovador que disminuye el stress a través de acariciar distintos elementos.

La clínica tiene osos de peluche, gatos, y retazos de alfombras.

Interfaces

Podemos identificar estas clases por ejemplo: **OsosDePeluche**, **Gato**, **Alfombra…**

Entonces, podría definir una **interfaz Acariciable** y que tenga como método:

+ acariciar();

Todos los elementos que implementen esta interfaz (que “respeten este contrato”) estarán obligados a implementar el método acariciar().

Interfaces

Entonces dentro de cada una de las clases, OsoDePeluche, Gato, Alfombra, **deberé implementar acariciar(); de cierta manera**.

Por ejemplo en **acariciar() del Gato, emitiré un ronroneo.**

Y si la clínica tiene una lista de **Acariciables**. Sabe que a todos los elementos de la lista, se los podrá **acariciar()**;

Interfaces

Como los elementos implementan la **interfaz Acariciable**, en el momento de la terapia, no nos interesará de qué Tipo son los elementos.

Los veremos como Acariciables...

No veremos: Gato, OsoDePeluche, Alfombra.

Veremos: Acariciable, Acariciable, Acariciable.

Y con certeza a todos les podremos invocar su método **acariciar()**.

UML - Interfaces

Para diagramar una interfaz con el diagrama de clases UML, la diagramamos como una clase común, que no tendrá atributos, y le agregamos **<<interfaz>>** sobre su nombre. Así:

UML - Interfaces

Gato

implementa (se comporta como)

Acariciable

En esta relación, decimos que **Gato** implementa **Acariciable**.
Se diagrama con una **línea punteada** y punta de flecha **vacía** desde el que implementa, hacia la interfaz.
Y sí volvemos a escribir el método **acariciar()** en **Gato**, porque lo deberá implementar.