**CLASES ABSTRACTAS**

Habrá ocasiones en las cuales necesitemos crear una clase madre donde únicamente coloquemos la estructura de una abstracción, una estructura muy general, dejando que sean las clases hijas quienes definan los detalles. En estos casos haremos uso de las clases abstractas.

Una clase abstracta es prácticamente idéntica a una clase convencional; las clases abstractas pueden poseer atributos, métodos, constructores, etc ... La principal diferencia entre una clase convencional y una clase abstracta es que la clase abstracta debe poseer por lo menos un método abstracto. pero ahora,

¿Qué es un método abstracto? un método abstracto no es más que un método vacío, un método el cual no posee cuerpo, por ende, no puede realizar ninguna acción. La utilidad de un método abstracto es definir qué se debe hacer, pero no el cómo se debe hacer.

Ejemplo

public abstract float area();

Si la clase tiene un método abstracto, se convierte en una clase abstracta.

**INTERFACES**

Ahora hablaremos de interfaces. Si hemos comprendido el tema de clases abstractas y el porqué de ellas, el tema de interfaces será un tema muy sencillo.

A diferencia de otros lenguajes de programación, en Java **no** es posible la herencia múltiple, nuestras clases únicamente podrán heredar de una y solo una clase.

Si conceptualizamos esto una representación pudiese ser la siguiente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El nivel de jerarquía es descendente.

Esto sin duda funciona, sin embargo, si queremos representar conceptos de la vida real necesitaremos una jerarquía mucho más compleja, algo como esto.

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Para que podamos diagramar nuestro proyecto de esta forma, teniendo en cuenta que únicamente es posible heredar de una clase, entonces haremos uso de interfaces.

**Podemos definir a una interfaz** como una colección de métodos abstractos y propiedades constantes en las que se especifica que se debe de hacer, pero no como, serán las clases hijas quienes definan el comportamiento.

A diferencia de una clase abstracta, una interface no puede hacer nada por sí sola, es prácticamente un contrato, en donde las clases que la implementen deben, obligatoriamente, definir el comportamiento de todos los métodos abstractos. ¿Cómo se debe hacer?

Veamos un ejemplo de una interfaz.

public interface ICanino

{

public abstract void aullar();

public abstract void ladrar();

}

Cómo podemos observar en la interfaz solo encontraremos métodos abstractos, método vacíos. Para poder implementar la interfaz basta con utilizar la palabra reservada implements.

public class Perro extends Mascotas implements ICanino {

Las cabeceras de los métodos declarados en el cuerpo de la interfaz se separan entre sí por caracteres de punto y coma y todos son declarados implícitamente como public y abstract (se pueden omitir). Por su parte, todas las constantes incluidas en una interfaz se declaran implícitamente como public, static y final (también se pueden omitir) y es necesario inicializarlas en la misma sentencia de declaración.

static -> indica que es accesible a nivel de Clase y no necesita una instancia de ella para ser usada.

final -> indica que el valor una vez declarado no puede ser cambiado. Y no pueden tener herencia.

public interface Constantes {

public static final double VALOR\_MAXIMO = 10000000.0;

double VALOR\_MINIMO = -0.01;

}

Las constantes que van en una interfaz son siempre con mayúsculas.

**En conclusión**, un método abstracto no es más que un método vacío el cual resuelve la pregunta ¿Qué se debe hacer? pero no el ¿Cómo se deba hacer? ya que serán las clases hijas quienes definan el comportamiento.

Los métodos abstractos podemos encontrarlos en dos entidades, las clases abstractas y las interfaces.

Una clase abstracta no es más que una clase común la cual posee atributos, métodos, constructores y por lo menos un método abstracto. Una clase abstracta no puede ser instanciada, solo heredada.

Cómo Java no permite la herencia múltiple habrá ocasiones en las cuales debamos utilizar interfaces, las cuales podemos verlas como contratos, contratos donde está muy bien establecido que debe hacer la clase que la implementa.