TEMA DE INVESTIGACION:

***Andrés Felipe Niño García***

INTERFACES:

Una **interfaz** en Java es una colección de métodos abstractos y propiedades. En las interfaces se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describan la lógica del comportamiento de los métodos.

La principal diferencia entre *interface* y *abstract* es que un interface proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia.

El uso de las interfaces Java proporciona las siguientes ventajas:

* Organizar la programación.
* permiten declarar constantes que van a estar disponibles para todas las clases que queramos (implementando esa interfaz)
* Obligar a que ciertas [clases](https://es.wikipedia.org/wiki/Clase_(programaci%C3%B3n)) utilicen los mismos [métodos](https://es.wikipedia.org/wiki/M%C3%A9todo_(programaci%C3%B3n)) (nombres y parámetros).
* Establecer relaciones entre clases que no estén relacionadas.

JAVA proporciona dos palabras reservadas para trabajar con interfaces: Interface e Implements:

* Para declarar una interfaz se utiliza:

modificador\_acceso **interface** **NombreInterfaz** {

....

}

Modificador\_acceso puede ser una clase de objetos que nos permite utilizar herencia en abstracción constante en las clases en las que se implemente.

* Para implementarla en una clase, se utiliza la forma:

modificador\_acceso **class** **NombreClase** **implements** NombreInterfaz1 [, NombreInterfaz2]

Una clase puede implementar varias interfaces de los paquetes que se han importado dentro del programa, separando los nombres por comas.

* Definición de una interfaz:

**interface** **Nave** {

**public** **abstract** moverPosicion (int x, int y);

**public** **abstract** disparar();

.....

}

* Uso de la interfaz definida:

**public** **class** **NaveJugador** **implements** Nave {

**public** void moverPosicion (int x, int y) {

*//Implementación del método*

posActualx = posActualx - x;

posActualy = posActualy - y;

}

**public** void disparar() {

*//Implementación del método*

}

...

# CLASE ABSTRACTA:

Hay ocasiones, cuando se desarrolla una jerarquía de clases en que algún comportamiento está presente en todas ellas pero se materializa de forma distinta para cada una. Por ejemplo, pensemos en una estructura de clases para manipular figuras geométricas. Podríamos pensar en tener una clase genérica, que podría llamarse FiguraGeometrica y una serie de clases que extienden a la anterior que podrían ser Circulo, Poligono, etc. Podría haber un método dibujar dado que sobre todas las figuras puede llevarse a cabo esta acción, pero las operaciones concretas para llevarla a cabo dependen del tipo de figura en concreto (de su clase). Por otra parte la acción dibujar no tiene sentido para la clase genérica FiguraGeometrica, porque esta clase representa una abstracción del conjunto de figuras posibles.

Para resolver esta problemática Java proporciona las clases y métodos abstractos. Un método abstracto es un método declarado en una clase para el cual esa clase no proporciona la implementación (el código). Una clase abstracta es una clase que tiene al menos un método abstracto. Una clase que extiende a una clase abstracta debe implementar los métodos abstractos (escribir el código) o bien volverlos a declarar como abstractos, con lo que ella misma se convierte también en clase abstracta.

* **Declaración e implementación de métodos abstractos**

Siguiendo con el ejemplo del apartado anterior, se puede escribir:

**abstract** **class** FiguraGeometrica {  
    . . .  
    **abstract** **void** dibujar();  
    . . .  
}  
  
**class** Circulo **extends** FiguraGeometrica {  
    . . .  
    **void** dibujar() {  
        // codigo para dibujar Circulo  
        . . .  
    }  
}

La clase abstracta se declara simplemente con el modificador *abstract* en su declaración. Los métodos abstractos se declaran también con el mismo modificador, declarando el método pero sin implementarlo (sin el bloque de código encerrado entre {}). La clase derivada se declara e implementa de forma normal, como cualquier otra. Sin embargo si no declara e implementa los métodos abstractos de la clase base (en el ejemplo el método dibujar) el compilador genera un error indicando que no se han implementado todos los métodos abstractos y que, o bien, se implementan, o bien se declara la clase abstracta.

**Referencias y objetos abstractos**

Se pueden crear referencias a clases abstractas como cualquier otra. No hay ningún problema en poner:

FiguraGeometrica figura;

Sin embargo una clase abstracta no se puede instanciar, es decir, no se pueden crear objetos de una clase abstracta. El compilador producirá un error si se intenta:

FiguraGeometrica figura = **new** FiguraGeometrica();

Esto es coherente dado que una clase abstracta no tiene completa su implementación y encaja bien con la idea de que algo abstracto no puede materializarse.  Sin embargo utilizando el up-casting visto en el capítulo dedicado a la Herencia si se puede escribir:

FiguraGeometrica figura = **new** Circulo(. . .);  
figura.dibujar();

La invocación al método dibujarse resolverá en tiempo de ejecución y la JVM llamará al método de la clase adecuada. En nuestro ejemplo se llamará al método dibujar de la clase Circulo.