***Diferencia entre una interfaz y una clase abstracta.***

**Herencia**

Vamos a empezar con una de las diferencias más conocidas entre una interfaz y una clase abstracta. Esta diferencia es sobre la herencia - cualquier clase puede implementar muchas interfaces, pero se puede extender una sola clase y puede tener un solo padre.

Clases múltiples que se extienden es una característica/funcionalidad que está presente solo en algunos lenguajes orientados a objetos. ¿Por qué? Porque trae más problemas que valor.

Cuando una clase tiene muchos padres y hay una situación que tenemos exactamente la declaración del mismo método en más de una, tendríamos que explícitamente "decir" cuál es el que nos interesa.

Dicho código es difícil de mantener, ya que hay que ir con cuidado a través de él cada vez que se está introduciendo cualquier cambio o refactorización. Por otro lado, si una clase tendría que extender (al menos) dos clases con el mismo método, entonces estamos hablando de violar la regla DRY y podemos extraer algo en otro lugar, o estamos perdiendo el Principio de Responsabilidad Única.

Bueno, estamos perdiendo el Principio de Responsabilidad Única si hay una necesidad de extender dos clases. Si cada uno es responsable de una cosa entonces algo que tiene que extenderse tanto de ellos es responsable de ... sí, creo que usted sabe lo que quiero decir.

"Vamos hombre, por lo que si una herencia de clase múltiple es tan malo, ¿por qué es bueno implementar muchas interfaces?" - Si una pregunta como ésta apareció en su mente tengo que admitir que es una maldita buena pregunta :)

Sin embargo, la respuesta es simple. Cada interfaz se basa en una funcionalidad en oposición a una clase - en la aplicación. Así que, incluso si estamos implementando diez interfaces de diferentes y cada uno contiene la declaración del mismo método que no hay colisión en ella. Interface es una garantía de la existencia del método, no la forma en cómo se tiene que implementar lo que significa que en todo momento no está violando el Principio de Responsabilidad Única usted debería estar bien con la implementación de muchas interfaces.

**Visibilidad del Método**

Todos los métodos de la interfaz son públicos, pero no hay tal regla para los declarados en las clases abstractas. Aquellos metodos pueden tener ninguna visibilidad excepto privado. ¿Por qué no privado? Debido a que un método abstracto debe implementarse en una subclase y si es privado no es accesible desde la subclase. Como podemos ver, estas dos ideas son mutuamente excluyentes.

Ok, pero vamos a volver al tema principal. Como escribí anteriormente, la interfaz es una garantía de una funcionalidad, usted puede tratarlo como un contrato entre las clases que está utilizando la interfaz y las clases que implementan la misma. Es una garantía de que una clase particular tendrá todos los métodos declarados implementados. Es por eso que esos métodos deben ser públicos. Cualquier otro no son tan importantes en este momento, porque están estrictamente relacionados con la aplicación.

Sin embargo, no es el caso cuando hablamos de clases abstractas. Siempre podemos tener un grupo de clases que son diferentes en tan sólo unos aspectos y salvo esto son casi lo mismo y el cuerpo de sus métodos públicos es también muy similar. Cuando nos encontramos en una situación como esta siempre podemos crear un método protegido que será el lugar donde se mantendrán las diferencias. Uno de los patrones mas populares que está utilizando este comportamiento es el Template Method.

**Declaración vs. Definición**

Una interfaz solo puede tener las declaraciones de los metodos y una clase abstracta puede o no contener la definición de los métodos.

Es así porque la interfaz se centra en proporcionar una funcionalidad específica y la clase abstracta puede enfocarse también en la similitud en las implementaciones de las subclases, no sólo por su funcionalidad.

**Constantes**

Tanto en las interfaces y en las clases abstractas no hay ningún problema con la definición de valores constantes. Es porque esos valores no depende de un objeto en particular, sino que son los mismos para todos.

**Atributos**

Las clases abstractas pueden contener atributos, mientras que las interfaces no pueden. La razón es la misma que con declaraciones y definiciones.