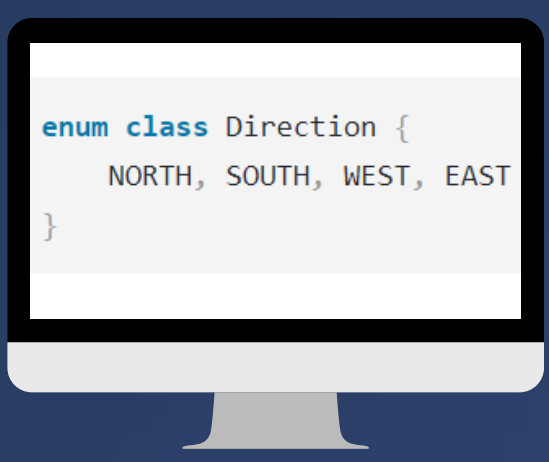
CONOCIENDO ACERCA DE: KOTLIN

¿Que es una Enumeracion ?

Una enumeración es un tipo definido por el usuario que consta de un conjunto de constantes enteras con nombre conocidas como enumeradores.

Las variables enumeradas, enumeraciones o más abreviadamente enum (palabra reservada), son un tipo especial de variables que tienen la propiedad de que su rango de valores es un conjunto de constantes enteras denominadas constantes de enumeración, a cada una de las cuales se le asigna un valor. Para mayor legibilidad del código estos valores del rango se identifican por nemónicos.

¿Cuál es Su Sintaxis en Kotlin?



¿Que es una Clase Sealed?

Las clases Sealed o comunmente llamadas Clases selladas representan jerarquías de clases restringidas que proporcionan más control sobre la herencia. Todas las subclases de una clase sellada se conocen en tiempo de compilación. No pueden aparecer otras subclases después de compilar un módulo con la clase sellada. Por ejemplo, los clientes de terceros no pueden extender su clase sellada en su código. Por lo tanto, cada instancia de una clase sellada tiene un tipo de un conjunto limitado que se conoce cuando se compila esta clase.

¿Son Enumeraciones con Superpoderes?

En cierto sentido, las clases selladas son similares a las enumclases : el conjunto de valores para un tipo de enumeración también está restringido, pero cada constante de enumeración existe solo como una instancia única , mientras que una subclase de una clase sellada puede tener varias instancias, cada una con su propia Expresar.
Una clase sellada es abstracta por sí misma, no se puede instanciar directamente y puede tener abstractmiembros.

Sintaxis de la Clase Sealead

Para declarar una clase sellada, coloque el sealed modificador antes de su nombre.

Result.kt

Users ▸ kamil.bekar ▸ Desktop ▸ Result.kt

```
sealed class Result<out R> {

    data class Success<out T>(val data: T) : Result<T>()
    data class Error(val exception: Exception) : Result<Nothing>()
    object Loading : Result<Nothing>()
}

val Result<*>.succeeded
    get() = this is Success && data != null

fun <T> Result<T>.successOr(fallback: T): T {
    return (this as? Success<T>)?.data ?: fallback
}

```

Expresion When:

El beneficio clave de usar clases selladas entra en juego cuando las usa en una when expresión . Si es posible verificar que la declaración cubre todos los casos, no es necesario agregar una else cláusula a la declaración. Sin embargo, esto solo funciona si lo usa when como una expresión (usando el resultado) y no como una declaración.

¿Que es El Patron de Diseño Singleston Y como funciona?

El patrón Singleton se centra en la creación de un objeto único, es decir, en vez de instanciar el mismo objeto muchas veces solo se instanciará una vez, y si ya ha sido instanciado se proporcionará un acceso al mismo.

Es útil para ya que muchas veces no nos interesará que haya dos objetos diferentes. Por ejemplo en ciertas ocasiones en nuestra aplicación tendremos diferentes usuarios, por lo que nos interesará que se puedan crear varias instancias de User para guardarlas en una base de datos. Pero y si nuestra apps solo necesita un usuario ya que su fin es simplemente guardar los datos en memoria del usuario que ha iniciado la sesión, el cual será siempre único, podríamos usar Singleton para evitar tener dos usuarios diferentes los cuales podrían provocarnos un error, ya que recordemos, son los datos de inicio de sesión y solo debería haber un usuario.

Ejemplo de un Programa en Kotlin con el uso de este Patron.

Singleton

```
public class Singleton {

    private static Singleton instance = null;

    private Singleton(){
    }

    private synchronized static void createInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
    }

    public static Singleton getInstance() {
        if (instance == null) createInstance();
        return instance;
    }

}
```