**UNIVERSIDAD DELAS REGIONES AUTÓNOMAS DE LACOSTA CARIBE NICARAGÜENSE**

**URACCAN: LAS MINAS**

CARRERA.

Ingeniería en sistema

NIVEL

IV año

ASIGNATURA.

Desarrollo de aplicaciones móviles I

DOCENTE.

Yesser miranda Guillen

AUTOR

Eliezer Iván Hernández Rivera

Fecha 29 Noviembre, 2020

UNIVERSIDAD DE LAS REGIONES AUTÓNOMAS DE LA

COSTA CARIBE NICARAGÜENSE

Ciencias, Tecnología y Medio Ambiente

Ingeniería en sistemas



GUIA 3: **PROGRAMACIÓN ORIENTADA A OBJETOS EN KOTLIN**

**ENUNCIADO**

# Clases básicas

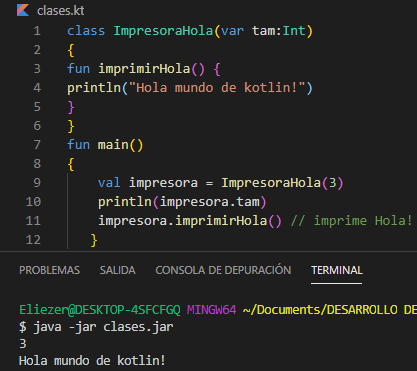
Existen dos pasos principales involucrados con las clases. Primero, debemos declarar nuestra clase, y luego podemos darle vida al instanciarla en un objeto utilizable real. Recuerde, la clase es solo un plano, y debe usar el plano para construir el objeto antes d poder hacer algo con él.

*  Abra su editor de texto preferido, es recomendable Visual Studio Code.
*  Cree una carpeta que contendrá los ficheros que vaya creando en esta práctica
*  Sincronice el proyecto con un repositorio de GitHub
*  Cree un fichero llamado Clases.kt y copie el siguiente código



Archivos compiladores de kotlin y archivo ejecutable de java.

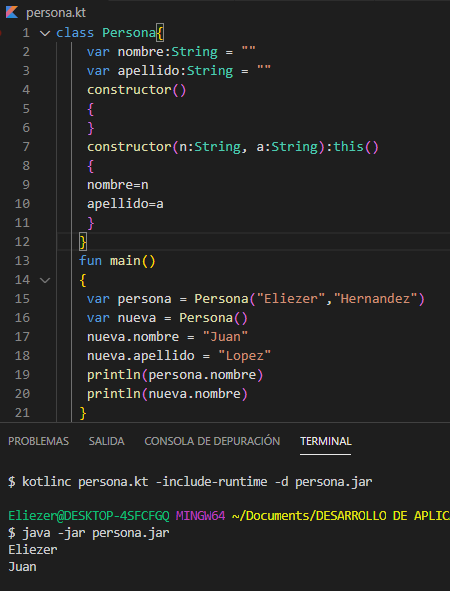
*  Compile y muestre el resultado. Confirme cambios en GIT



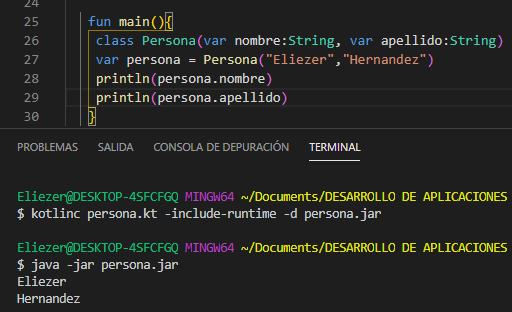
*  Los constructores en kotlin se pueden declarar usando la palabra clave *constructor* o se puede omitir si se establecen las propiedades a lado del nombre de la clase. Copie el siguiente código en un fichero llamado Persona.kt



*  Imprima los resultados e indique que operación realizamos. confirme cambios en GIT.



*  Cambie la estructura de la clase anterior a la siguiente y luego confirme en GIT



*  Utilizando el código del main anterior indique que operaciones no se permiten usando esa estructura de clase, tomando en cuenta que utiliza el constructor directamente al lado del nombre de la clase. Comente el código no permitido.

El motivo de cambiar la clase está en que la estructura dela clase persona hace instancia a una sola persona y no hace referencia a ningún constructor ni vacío.

*  Ejercicio: Cree una lista de personas y recorra la lista usando un for, agregue 3 personas y muestre los resultados de los datos de cada persona:

var lista\_personas = ArrayList<Persona>()

# 

# Objetos, data class y class enum

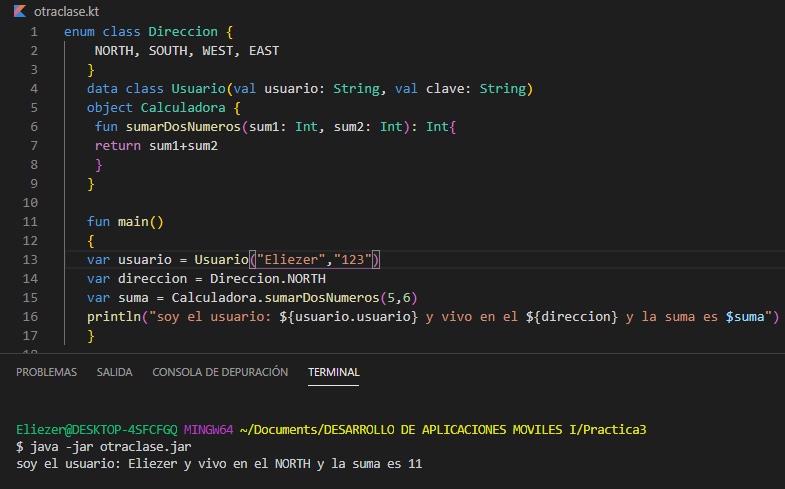
Kotlin facilita algunas operaciones mediante objetos, clases de datos y enum. En el caso de los objetos es para facilitar la creación de singleton en una aplicación. Las clases de datos para definir una clase sin funcionalidad y las enum class para enumeraciones.

Podríamos indicar que los object de Kotlin, se asemejan a las clases estáticas de otros lenguajes

*  Cree un nuevo fichero llamado OtrasClases.kt dentro de la misma carpeta



*  Copie el siguiente código sobre enumeraciones, clases de datos y objetos



 Analice la utilización de cada uno imprimiendo su resultado

Los datos de este ejercicio muestran lo siguiente

# Herencia y polimorfismo

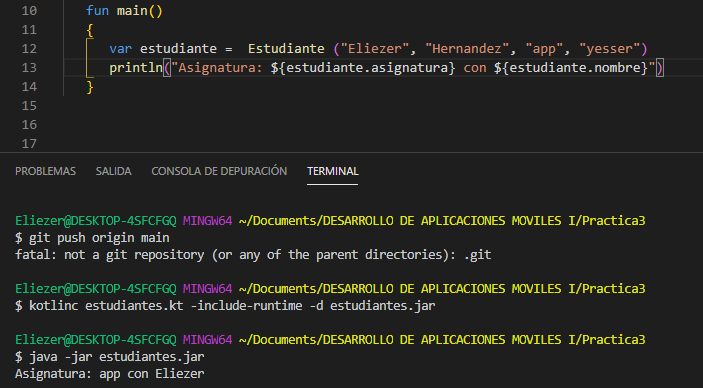
Un programa usa el flujo básico de ejecución, de esta manera si no controlamos que código queremos que se ejecute, entonces, se ejecutará todo de arriba hacia abajo y de izquierda a derecha.

Utilizaremos la clase Persona con las últimas ediciones indicadas para utilizarla como clase base para demostrar el uso de herencia de clases. Para poder utilizar herencia de clases es necesario preceder la palabra clave **open.** Si no indica con open y aplica herencia kotlin no se lo va a permitir.

*  Cree un nuevo fichero llamado Estudiante.kt

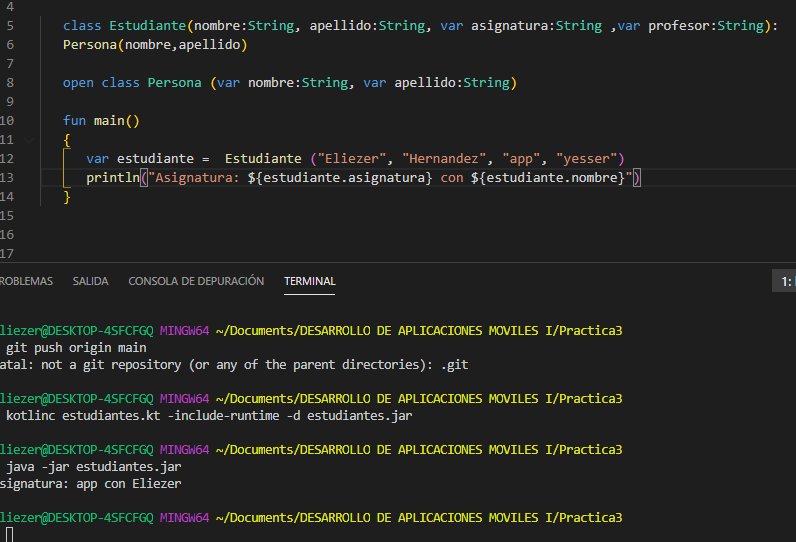


*  Copie el código a continuación sobre la clase Estudiante que herede de la clase Persona

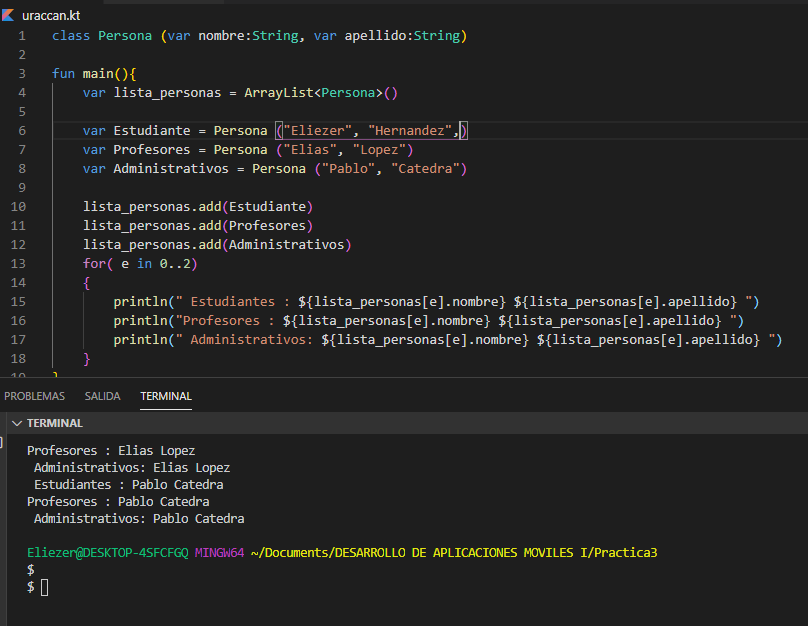
 Antes de probar el resultado recuerde que debe agregar la palabra clave **open** a la clase Persona

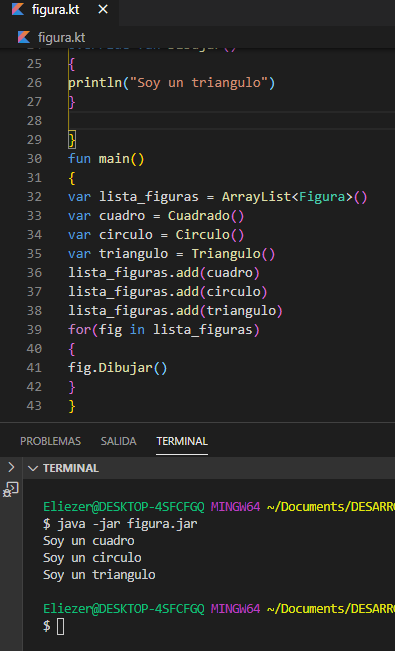
 Al final puedes comprobar que puedes imprimir la propiedad nombre desde un objeto estudiante, para eso, en el main construye un objeto estudiante con los cuatro parámetros que se piden en su constructor.

 Luego de construir el objeto, puedes imprimir el siguiente mensaje.



 Analice bien el resultado, tomando en cuenta que la propiedad nombre pertenece a Persona y no a estudiante.

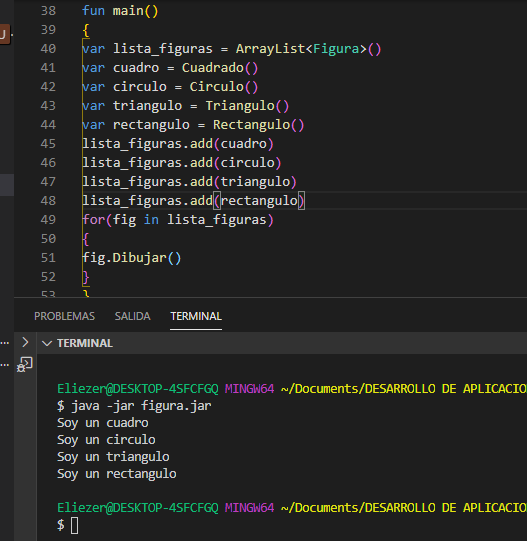
* Si es cierto que nombre pertenece a la open pero en una clase es lo opuesto porque permite que otros hereden de esta clase. Por defecto, todas las clases en Kotlin son final, lo que corresponde que hereda de OPEN.
*  **Ejercicio:** La universidad URACCAN, requiere de un control de personas en su recinto, de la universidad únicamente se almacenará el recinto y dirección. Teniendo en cuenta que en la universidad encontramos **Estudiantes, Profesores y administrativos**. Los datos en común de todos ellos que se pueden declarar en persona son: **nombre, apellido, cedula y edad**. De los estudiantes se requiere el nivel educativo y el nombre de su profesor representante. De los profesores se requiere: facultad y cantidad de asignaturas. De los administrativos se requiere: área de trabajo y nombre del jefe inmediato. Debe tener en cuenta que se debe crear una lista de personas que pueden ser Estudiantes, Profesores y Administrativos.
* 
*  Es posible que el ejercicio anterior necesite más información y ejemplos, para eso crearemos una clase figura con la palabra clave open
*  Luego de haber creado la clase anterior agregue el código para una clase Cuadrado que herede de figura y que cancele con override el método dibujar de su clase padre.
*  De la misma manera agregue dos clases más una de Circulo y otra de Triangulo que hereden de Figura. Cada una con un mensaje personalizado.
*  Ahora solo nos queda probar en la función main



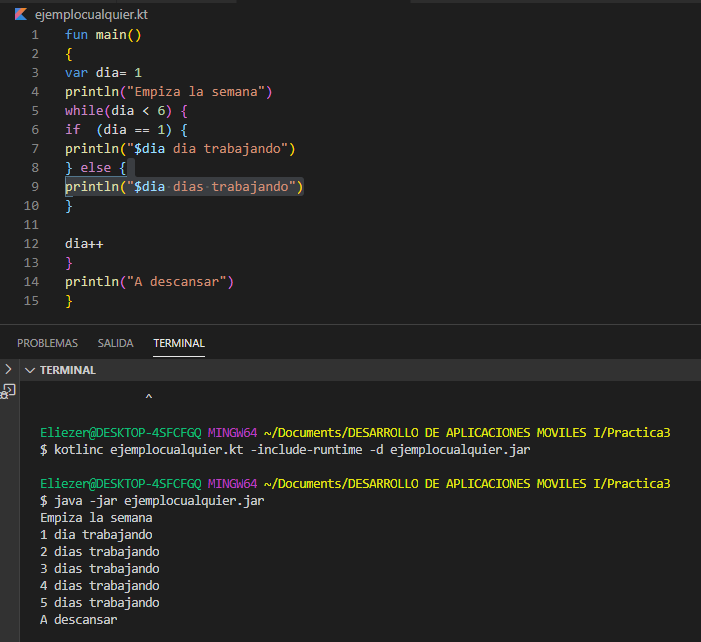
 Analice bien el resultado, confirme cambios en GIT. Observe que se hace una lista de figuras pero se agregan objetos de otro tipo, entonces, indique porque se nos permite esto.

Esto lo permite porque le agrego la palabra Open a cual es heredada por otras clases.

 **Ejercicio:** En el programa anterior, agregue una funcionalidad más, que se permita ingresar un nombre a través del teclado y que utilice el método find de las listas para encontrar una coincidencia, es necesario el uso de una variable nulable.



**Investigue**:

1. Realice cualquiera de los ejemplos realizados, pero esta vez utilice modificadores de visibilidad (internal public, protected y private ). Tenga en cuenta que el modificador por defecto es public.
2. 
3. Implementa un ejemplo de desestructuración en Kotlin, que declare dos variables y se asignen a través del teclado usando desestructuración.
4. Clases sealed vs clases abstractas.

Las clases sealed se utilizan para una jerarquía limitada y cerrada de casos posibles para ciertos tipos de datos.

La principal limitación de las clases abstractas es que dejan de tener un conjunto limitado de valores, puesto que se podría crear una clase que herede de otra clase en cualquier otra parte de un código, por lo cual perdemos el control sobre los subtipos posibles. Las sealed classes vienen a cubrir este hueco que hay entre los enumerados y las clases abstractas.

1. Implemente un ejemplo usando interfaces en POO