Estudiante: Andres Baldiviezo Farfan **Paralelo: “A”**

APUNTES

KOTLIN

Kotlin es un lenguaje de programación que está diseñado para programación moderna y es mas atractivo, y trabaja similarmente a java de hecho se ejecuta en la maquina virtual de java es decir la JVM en nuestra materia kotlin nos servirá para la programación móvil e introducirnos a este campo.

Kotlin ofrece muchas características modernas que lo hacen atractivo para los desarrolladores, como la seguridad de tipos, la inferencia de tipos, las funciones de orden superior, las extensiones de funciones, las clases de datos, la programación funcional y la concisión del código. Estas características pueden ayudar a aumentar la productividad y mejorar la legibilidad del código.

Su sintaxis concisa, su interoperabilidad con Java y su amplia gama de características lo convierten en una opción popular entre los desarrolladores de software.

FUNDAMENTOS - KOTLIN

- Kotlin está basado en Java.
- Comparten la JVM Java Virtual Machine.
- javac – kotlinc {bytecode}
- Interoperabilidad.

Variables

Es un contenedor que puede almacenar un valor.

Sus características son:

- Tiene un nombre.
- Un tipado.
- ¿Puede reutilizarse? Tenemos dos formas:

Val : No reutilizable. Es decir una constante.

Var : reutilizable.

Variables mutables: Una variable mutable es aquella cuyo valor puede ser modificado después de su creación. Esto significa que puedes cambiar el contenido o el estado de la variable durante la ejecución del programa. Ej:

var x = 15

x = 5

x= 11

este puede cambiar sin problemas ni errores

Variables inmutables: Una variable inmutable es aquella cuyo valor no puede ser modificado una vez que ha sido asignado. Esto significa que una vez que se ha creado la variable con un valor inicial, no se puede cambiar ese valor. Ej:

val x= 15

x= 15 // esta línea dará un error

por que es inmutable y no puede cambiar

un buena práctica que podemos utilizar es escribir las variables inmutables con mayúscula por ejemplo:

val PI = 3.1416

val SUMA = 15;

PROGRAMACION FUNCIONAL

La programación funcional es un paradigma de programación que se centra en el uso de funciones y en el tratamiento de la computación como la evaluación de funciones matemáticas. Se basa en el concepto de funciones puras, que son funciones que, dada la misma entrada, siempre producen el mismo resultado y no tienen efectos secundarios observables.

Funciones de primera clase

En la programación funcional, las funciones se tratan como ciudadanos de primera clase, lo que significa que pueden ser pasadas como argumentos a otras funciones, devueltas como valores de otras funciones y asignadas a variables.

Funciones puras

Como se mencionó anteriormente, las funciones puras no tienen efectos secundarios y producen el mismo resultado cuando se les pasa la misma entrada. Esto hace que el código sea más predecible, más fácil de razonar y menos propenso a errores.

Inmutabilidad

La mayoría de los valores en la programación funcional son inmutables, lo que significa que una vez que se crea un valor, no se puede cambiar. Esto evita los efectos secundarios y simplifica la gestión del estado.

Recursividad

La recursión se utiliza comúnmente en la programación funcional en lugar de bucles iterativos. Las funciones recursivas se llaman a sí mismas con argumentos modificados hasta que se alcanza un caso base.

Programación declarativa

La programación funcional tiende a ser más declarativa que imperativa. En lugar de especificar paso a paso cómo lograr un resultado, te centras en describir qué quieres lograr.

Evaluación perezosa

En algunos lenguajes funcionales, la evaluación perezosa (o diferida) se utiliza para evitar el cálculo innecesario. Las expresiones se evalúan solo cuando su valor es necesario.

PROGRAMACION ORIENTADA A OBJETOS

Objeto nombrado:

Uso de la palabra clave ‘object’ para definir un objeto sin necesidad de declarar una clase

Clase: Plantilla para definir propiedades y métodos

Método: Entidad independiente con sus propios datos y programación

Definir un método:

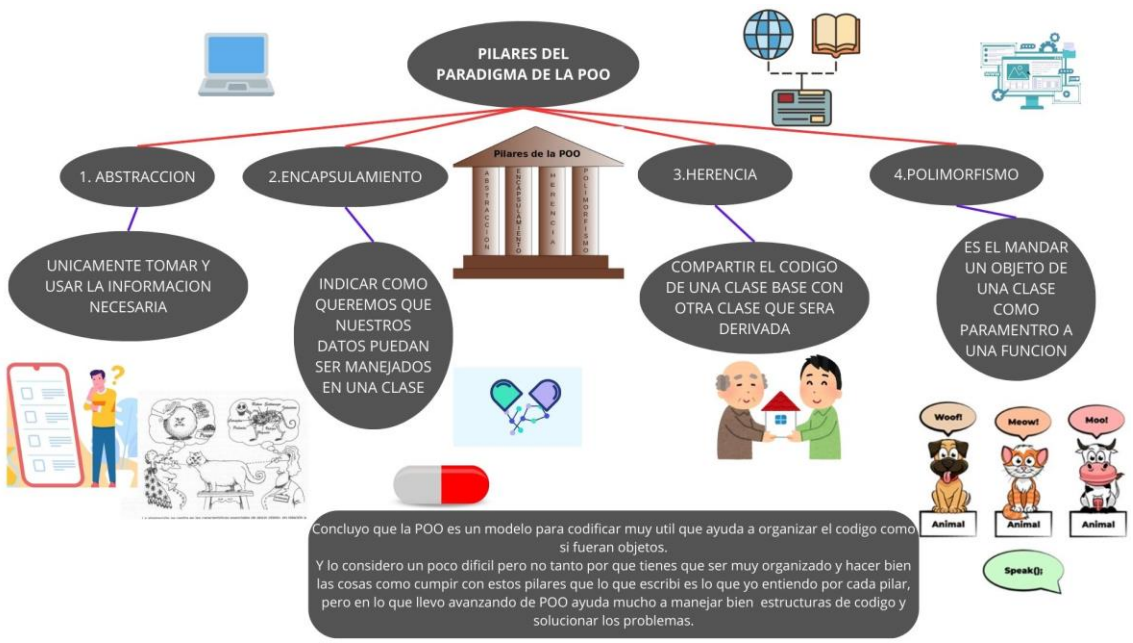
fun imprimir() {

println("Nombre: $nombre y tiene una edad de $edad")

}

Método constructor:

class Persona (var nombre: String, var edad: Int)



ABSTRACCION

Es la capacidad de representar los elementos esenciales de un objeto sin necesidad de incluir todos los detalles de implementación. En POO, los objetos son entidades que tienen atributos y métodos asociados. La abstracción permite modelar objetos de manera que se centren en lo que son capaces de hacer y en cómo interactúan con otros objetos, más que en los detalles internos de cómo se implementan esos métodos.

Ejemplo: una clase animal tiene métodos como hacerSonido() y moverse() pero no especifica como lo hace para un animal especifico como un perro o un gato, pero ambos siguen siendo de tipo animal

Encapsulamiento:

Es la ocultación de los detalles de implementación de un objeto y la protección de sus datos internos. En términos más simples, encapsulamiento significa que los datos de un objeto solo pueden ser accedidos y modificados a través de métodos específicos definidos para ese objeto.

public: Los miembros marcados como públicos son accesibles desde cualquier parte del programa. No hay restricciones en su acceso.

private: Los miembros marcados como privados solo son accesibles dentro de la misma clase. No pueden ser accedidos directamente desde fuera de la clase.

protected: Los miembros marcados como protegidos son accesibles dentro de la misma clase y por las clases derivadas (subclases), pero no desde fuera de ellas.

Herencia:

Es la que permite la creación de nuevas clases basadas en clases que ya existen, lo que facilita la reutilización de código y la creación de jerarquías de clases.

En la herencia, una clase (subclase) hereda atributos y métodos de otra clase (clase base, superclase o padre). La clase derivada puede agregar nuevos miembros, modificar el comportamiento de los miembros heredados y proporcionar nuevos métodos y atributos específicos si es necesario.

Polimorfismo:

Este permite que un mismo método tenga diferentes comportamientos según el tipo de objeto que lo esté invocando. El polimorfismo se puede lograr mediante el uso de la herencia y los métodos de sobreescritura.

Por ejemplo constructores que suelen ser iguales pero reciben diferentes parámetros (estático)

Y los métodos heredados que se pueden sobrescribir o añadirle mas líneas de código necesarias(dinamico)

Ejemplo:

```
open class Animal {
   open fun hacerSonido() {
      println("El animal hace un sonido")
   }
}
```

```
class Perro : Animal() {
    override fun hacerSonido() {
        println("El perro ladra")
    }
}
class Gato : Animal() {
    override fun hacerSonido() {
        println("El gato maulla")
    }
}
```

Ambos heredan de animal con el mismo contexto pero sobrescriben sobre el método para tener su propia implementación