BECAS SANTANDER TECNOLOGÍA | CARRERAS DIGITALES | BEDU

PROYECTO FIN DE MÓDULO.

“SuperTianguis”

DESARROLLO MÓVIL ANDROID MÓDULO 2. KOTLIN FUNDAMENTALS

EQUIPO 13:

MARCO ANTONIO FLORES PEREZ
marcopflores@yahoo.com.mx

GUILLERMO AGUSTÍN DZIB DURÁN
guillermo.add@hotmail.com

ARTURO RIOS CUETO
arturo.rioscueto@gmail.co

JOSE RAMÓN ROJAS PÉREZ
rojas9987cb12@gmail.com

JOSÉ GARCÍA
jose080715@gmail.com

REPOSITORIO DEL PROYECTO:

https://github.com/ArturRiosDev/Proyecto_kotlin_fase2

VIDEO DEL PROYECTO:

https://drive.google.com/file/d/1VabNVi_PsWXp8cJyxXeS6YVzv9qVArxy/view?usp=sharing

Contenido

PROYECTO KOTLIN FUNDAMENTALS

superTianguis

DEFINICIÓN DEL PROYECTO

Este proyecto se basará en las distintas aplicaciones de marketplace ya existentes, como lo son mercadolibre, amazon, Marketplace etc. para quienes inician un negocio o están en búsqueda de una alternativa que les dé mayor difusión de sus productos por medio de una aplicación web, esta app solucionan la necesidad de tener su propia tienda en línea, además de que sea seguro, garantice ser económicamente accesible y confiable al cliente. El código del proyecto puede consultarse en el siguiente repositorio y clonarse para correrlo en la consola:

https://github.com/ArturRiosDev/Proyecto_kotlin_fase2

ALCANCE DEL PROYECTO

Este proyecto no tiene el objetivo de convertirse en una aplicación comercializable, de hecho, solo tiene la finalidad de servir como herramienta de práctica para reforzar todos los conocimientos adquiridos durante el módulo de fase II, por medio de una aplicación que simula a una aplicación real, probar distintas formas de resolver problemas, practicar y mejorar habilidades de programación. Por lo anterior, no es obligatorio crear una aplicación desde cero junto a su concepto y funcionalidades, sino que tomar de una ya existente todo lo anterior y así encontrar la forma de simular todas sus funcionalidades dentro de un código que se ejecute de forma similar.

REQUERIMIENTOS

- Escritura y estructura del código.
- Implementación de programación orientada a objetos.
- Definición del proyecto.
- Cobertura de los temas.
- Flujo del proyecto.

FLUJO DE LA APP

Menú inicial

En pantalla inicial se presenta un menú de opciones para registrar un nuevo usuario, para iniciar sesión con un usuario existente, una lista de usuarios registrados y para cerrar la app (Imagen 1).

La selección se hace mediante una entrada de datos de números enteros, si se inserta un dato que no aparece, una cadena de texto, un carácter, un número con punto decimal o cualquier otro tipo que no sea un entero la aplicación le dirá al usuario que solo se aceptan números enteros, reimprimirá todo el menú inicial y solicitará nuevamente un dato entero.

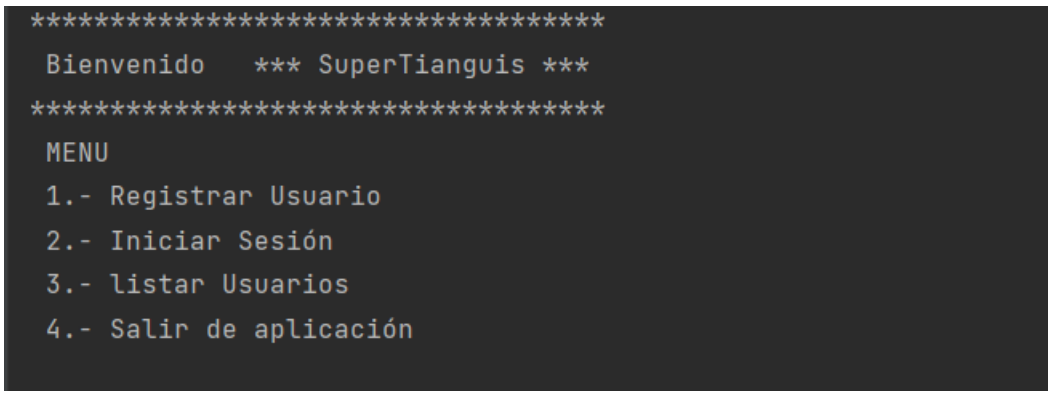


imagen 1 Menú inicial

Registrar usuario

Al seleccionar la opción 1 del menú de inicio (imagen 1), inicia un proceso de registro de usuario nuevo, por lo que se solicita que se ingrese el nombre de usuario nuevo y contraseña. La entrada de datos acepta cualquier tipo de datos y el registro se guarda como cadenas de texto (imagen 2).

Al terminar el registro se regresa automáticamente al menú de inicio.

```
1
Registrar usuario nuevo
 Introduce  usuario: guille
 Introduce password: 666
Usuario creado con exito
```

imagen 2 Registro de usuario

Iniciar sesión

Al seleccionar la opción 2 del menú de inicio (imagen 1), se ejecuta el inicio de sesión que consiste básicamente en la solicitud de usuario y contraseña. De igual forma acepta cualquier tipo de dato y lo recibe como una cadena de caracteres, con la que hace una comprobación de si existe en la data (un mutableList) (imagen 3).

Al terminar el procedimiento se pasa automáticamente al menú de selección de productos (imagen 5).

```
2
 Iniciar Sesión
 Introduce cuenta usuario: guille
 Introduce password: 666
```

imagen 3 Inicio de sesión

Listar usuarios

Al seleccionar la opción 3 del menú de inicio (imagen 1), se ejecuta un simulacro de descarga de usuarios, en donde se muestran todos los usuarios contenidos en la lista, si anteriormente se registró un usuario nuevo, aparecerá ahí (imagen 4).

Automáticamente posterior a eso se reimprimirá nuevamente el menú de inicio (imagen 1).

```
3
Descargando usuarios.....
marco
juan
angie.
guille
El venado
```

imagen 4 Listado de usuarios existentes

Listado de productos

Al concluir el llenado del formulario del inicio de sesión (imagen 3), se enlistará automáticamente un menú de opciones con todos los productos que están a la venta (imagen 5). La manera de seleccionar el producto para agregarlo al carrito es ingresando el número “SKU” del producto. Esta entrada solo permite números enteros, si se le ingresa una cadena de texto o cualquier otro tipo de dato, lanzará una excepción controlada con un mensaje de “El producto que buscas no existe”. Y solicitará nuevamente un número entero.

Al ingresar una entrada válida se preguntará cuántos desea comprar (imagen 6) y nuevamente se hará una validación del tipo de dato, si no es entero se volverá a iniciar el menú de listado de productos (imagen 5).

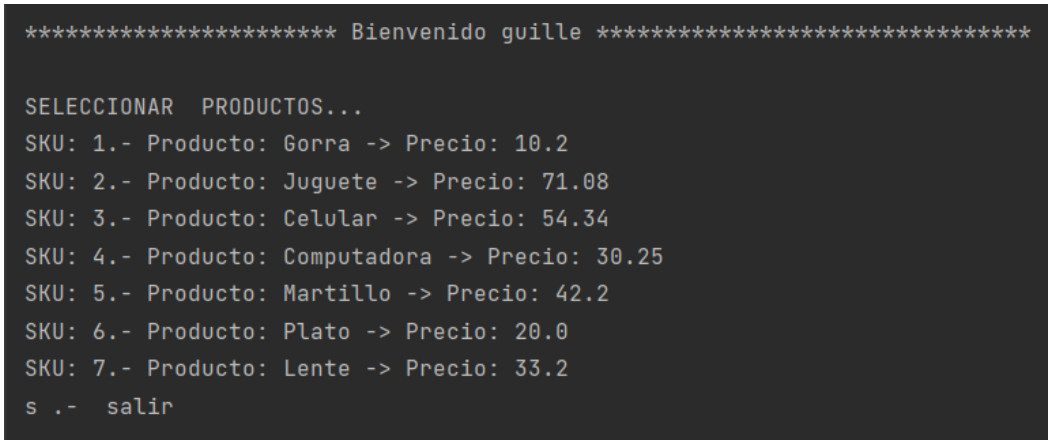


imagen 5 Menú de listado de productos

Selección de productos

Al seleccionar un producto, como se explicó en el apartado de listado de datos (imagen 5), se pregunta la cantidad a comprar, después de la validación del tipo de dato, se imprime el carrito de compras actualizado, con el producto seleccionado. Posteriormente se imprime nuevamente una instrucción con dos opciones, “seguir comprando” o “realizar la compra“(imagen 6).

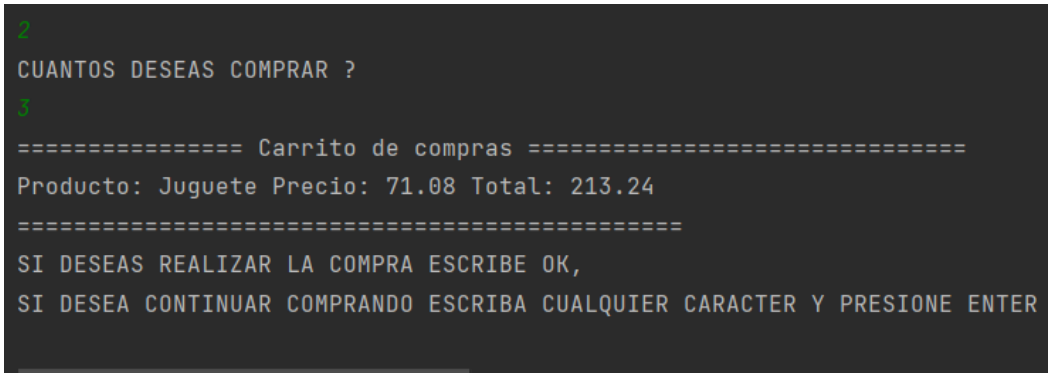


imagen 6 Procedimiento de selección de productos

Seguir comprando

Para seguir comprando se debe ingresar cualquier entrada de datos, excepto la palabra “ok”, y se mostrará nuevamente el listado de productos para seguir comprando (imagen 7). Y se podrá repetir el proceso anterior, ingresando cuantas veces sean necesarias productos al carrito hasta ingresar “ok”.

```
SI DESEAS REALIZAR LA COMPRA ESCRIBE OK,
SI DESEA CONTINUAR COMPRANDO ESCRIBA CUALQUIER CARACTER Y PRESIONE ENTER
f

SELECCIONAR  PRODUCTOS...
SKU: 1.- Producto: Gorra -> Precio: 10.2
SKU: 2.- Producto: Juguete -> Precio: 71.08
SKU: 3.- Producto: Celular -> Precio: 54.34
SKU: 4.- Producto: Computadora -> Precio: 30.25
SKU: 5.- Producto: Martillo -> Precio: 42.2
SKU: 6.- Producto: Plato -> Precio: 20.0
SKU: 7.- Producto: Lente -> Precio: 33.2
s .-  salir
5
CUANTOS DESEAS COMPRAR ?
2
================ Carrito de compras ==============================
Producto: Juguete Precio: 71.08 Total: 142.16
Producto: Martillo Precio: 42.2 Total: 84.40
=================================================
SI DESEAS REALIZAR LA COMPRA ESCRIBE OK,
SI DESEA CONTINUAR COMPRANDO ESCRIBA CUALQUIER CARACTER Y PRESIONE ENTER
```

imagen 7 Procedimiento para seguir comprando después de haber seleccionado un producto

Realizar compra

Para realizar la compra con los productos seleccionados hasta el momento, se debe ingresar “ok” sin importar si está en mayúsculas o minúsculas. Y saldrá la impresión ”compra realizada con éxito” (imagen 8). Al ocurrir esto pasan dos cosas, la primera es que se borra todo lo contenido en la lista del carrito de comporas y lo segundo es que se regresará nuevamente al menú de inicio para que otro usuario pueda iniciar sesión a la tienda.

```
SI DESEAS REALIZAR LA COMPRA ESCRIBE OK,
SI DESEA CONTINUAR COMPRANDO ESCRIBA CUALQUIER CARACTER Y PRESIONE ENTER
ok
compra realizada con exito

***********************************
 Bienvenido   *** SuperTianguis ***
***********************************
 MENU
 1.- Registrar Usuario
 2.- Iniciar Sesión
 3.- listar Usuarios
 4.- Salir de aplicación
```

imagen 8 Proceso de finalizar la compra de todos los productos agregados al carrito

LÓGICA DEL PROYECTO CON DIAGRAMAS DE FLUJO

A continuación se presentan los diagramas de flujo de la lógica del proyecto, cada diagrama de flujo representa una clase y contiene una explicación del mismo.

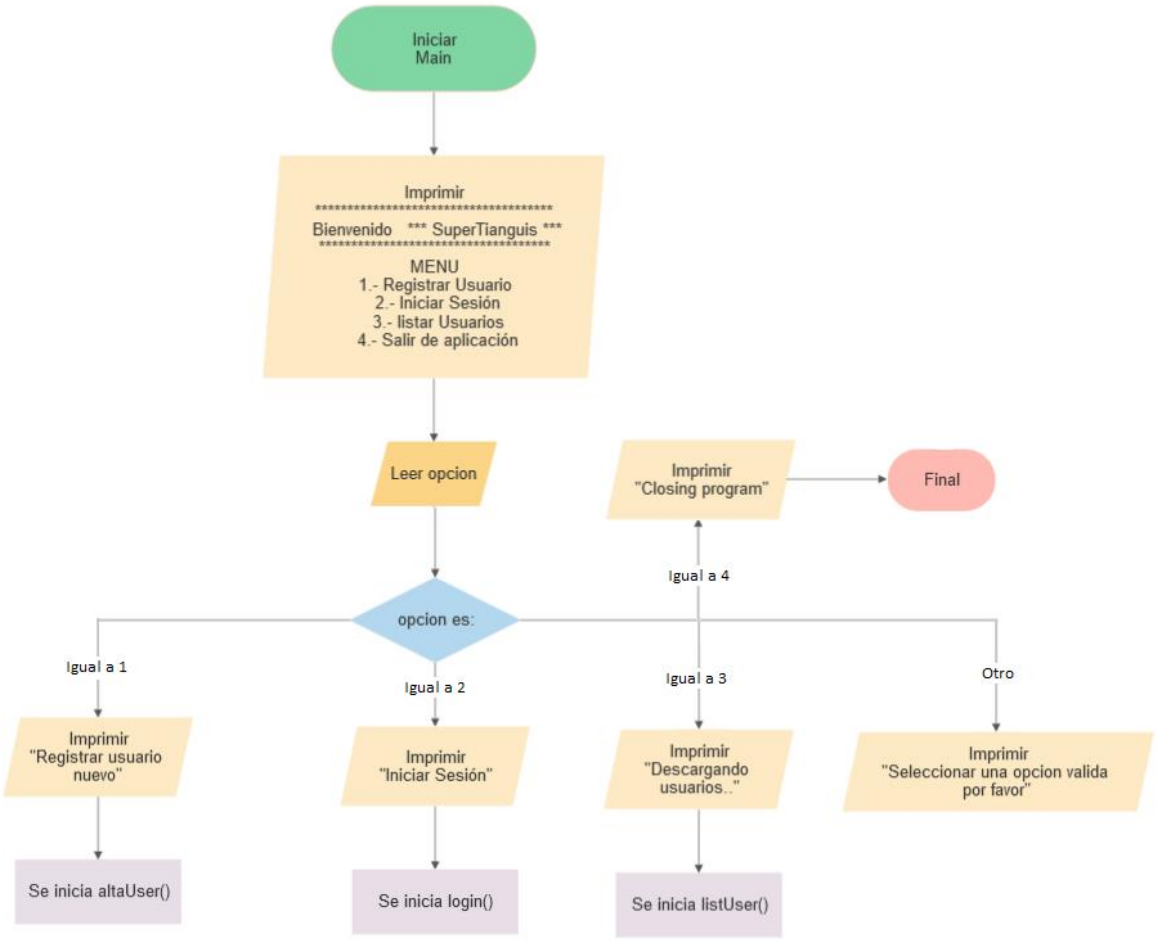


Diagrama de flujo 1 Clase Main

La clase Main (Diagrama de flujo 1) contiene la lógica del menú inicial (imagen 1), consiste únicamente en la entrada de datos para elegir una opción y al ser seleccionada, ejecuta el método que es una instancia de la clase UserManager() (Diagrama de flujo 2)

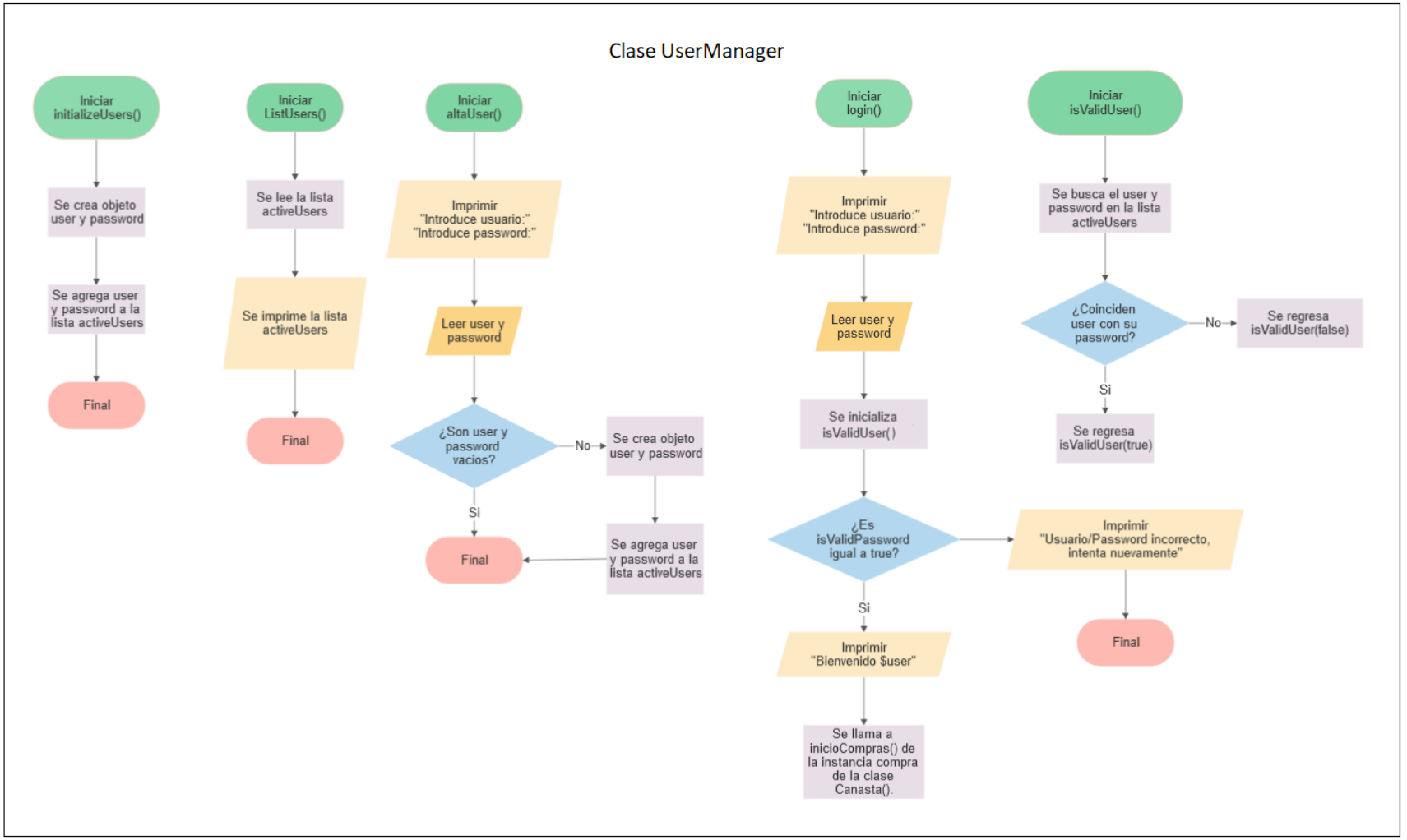


Diagrama de flujo 2 Clase UserManager y sus funciones

La clase UserManager (Diagrama de flujo 2), es una clase hija de la clase abstracta User (Diagrama de flujo 3). Contiene la lógica de los métodos para la selección de la clase Main (Diagrama de flujo 1). Al crear una instancia de la Clase UserManager, se ejecuta automáticamente el método initializaeUsers(), el cual tiene como misión llenar la lista de usuarios (lista activeUsers) que sirve como un emulador de la base de datos de los usuarios que utiliza nuestro programa.

La clase también contiene el método listUsers() que tiene como función imprimir la lista de usuarios registrados (lista activeUsers).

Contiene el método altaUser() que es el encargado de dar de alta usuarios nuevos, consiste en una petición de entrada de datos el cual tiene una verificación de nulabilidad y tipo de datos válida. Al final acondiciona y agrega el dato registrado a la lista de usuarios (lista activeUsers).

Por último, los métodos login() y isValidUser() están relacionados entre sí. Siendo el primero el encargado de solicitar una entrada de datos para el usuario y contraseña, verifica nulabilidad, tipo de datos válido, si el usuario existe en la lista de usuarios () y si la contraseña de datos coincide con el asignado al usuario ingresado. Esto último con el método isValidUser que regresa un Booleano si dicha condición se cumple o no. Al final, si estas condiciones se cumplen se ejecuta el método inicioCompras() que es instancia de la clase Canasta.

Diagrama de flujo 3 Clase User

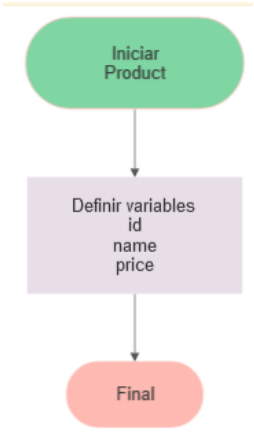


Diagrama de flujo 4 Clase Product

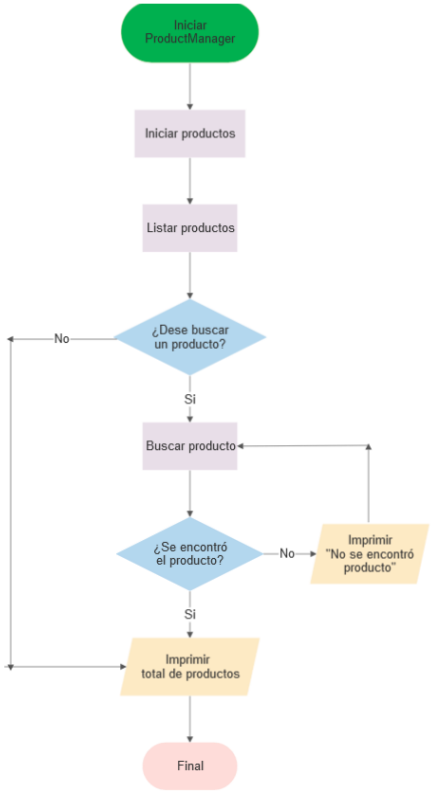


Diagrama de flujo 5 Clase ProductManager

La clase user (Diagrama de flujo 3) se trata solo de una clase abstracta, la cual le dice a sus clases hijas que deben implementar el método listUsers(). En este caso su clase hija es la clase UserManager (Diagrama de flujo 2)

La clase Product (Diagrama de flujo 4) se trata solo de una data class, que tiene la función de definir las los atributos de un producto, en este caso el id, el name y el price

La clase ProductManager (Diagrama de flujo 5) hace más de una función la primera es que al instanciarse, automáticamente crea la lista de productos (lista productos) y le agrega elememento, esta lista simula ser la base de datos de productos del proyecto. La clase incluye un método para la impresión de la lista de productos (método listProducts()), un método para imprimir el total de la compra según la cantidad seleccionada cuando se haga la compra (método totalProducts()) y un método para buscar productos en la lista de productos (método findProducts())

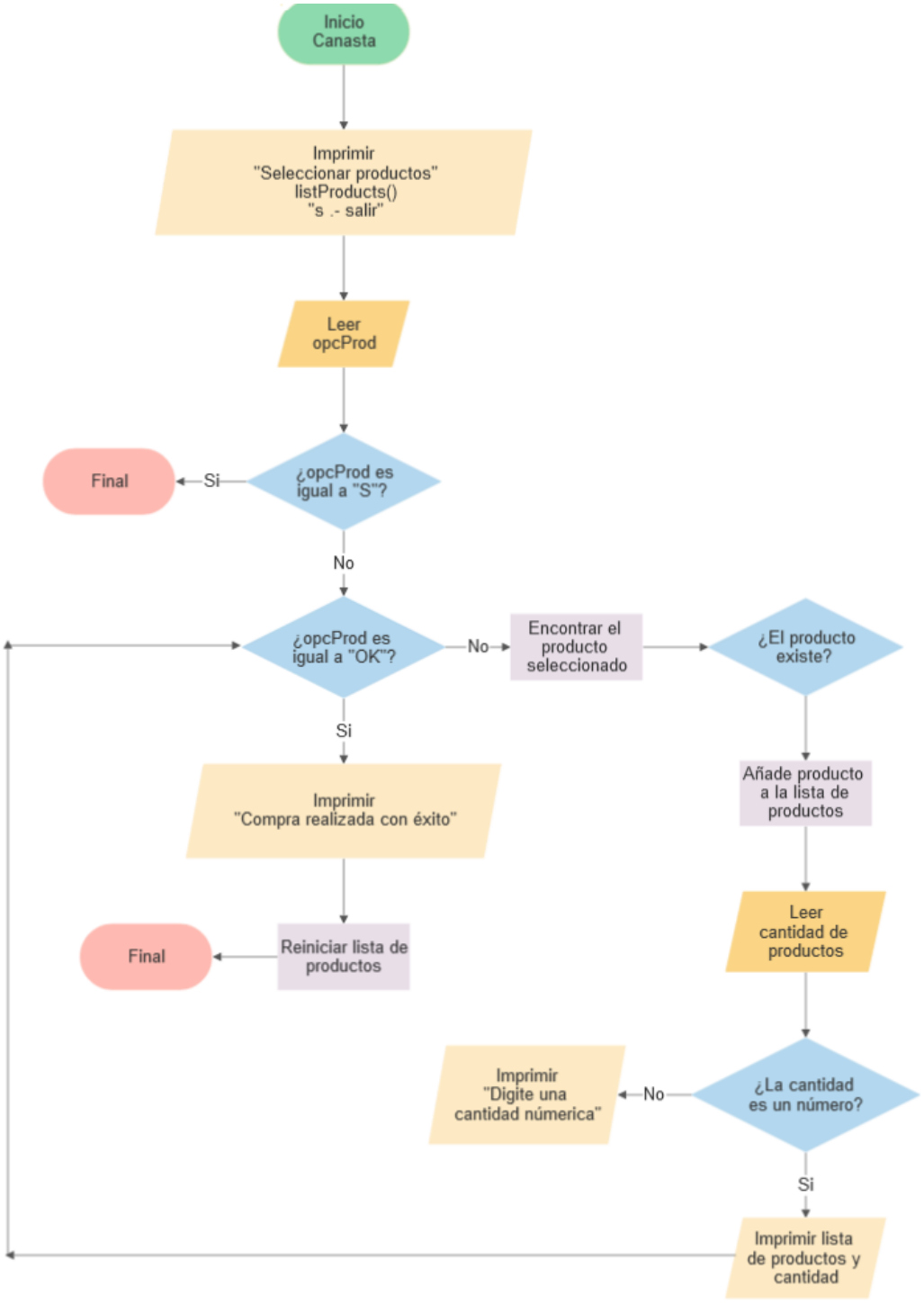


Diagrama de flujo 6 Clase canasta

La clase Canasta administra la lógica de la compra (Diagrama de flujo 6), imprime la lista de productos y espera una entrada de datos (variable opcProd), cuenta con todas las verificaciones de tipo de datos en todos los pasos y al ingresar el SKU verifica si el producto existe y si es así lo agrega a la lista del carrito (lista listOfProduct), posteriormente pregunta la cantidad de productos a comprar y al ingresarlo, imprime el carrito de compras

y el método totalProducts() que es una instancia de la clase ProductManager. Por último imprimirá una pregunta de si se desea seguir comprando o se desea finalizar la compra, si se ingresa “ok” la compra se realizará, si por el contrario se ingresa cualquier otro carácter o cadena de caracteres se seguirá comprando regresando al inicio del flujo.

PROYECTO VS TODOS LOS TEMAS VISTOS EN EL CURSO

A continuación se hace un listado de todos los temas vistos en el curso y su implementación en el proyecto.

Variables , tipos de datos

Se definieron, inicializaron y utilizaron variables mutables e inmutables, de diferentes tipos de datos, según su utilidad.

```
public String name;
1 usage
public String password;
```

```
(var id: String, var name: String, var price: Double)
```

```
val cantidad: Int = readLine()!!.toInt()
```

Operadores

Se utilizaron operadores para el manejo de algunas tomas de decisión u operaciones requeridas.

```
opcProd == "OK"
```

```
productoAComprar == false
```

```
(id.toInt() > productos.size || id.toInt() < 1)
```

```
(productoEncontrado != null)
```

Funciones

Las funciones son necesarias en cualquier tipo de proyecto, algunas de las utilizadas en el presente son las siguientes.

```
fun findProduct(id: String): Any? {
    try {
        val productoEncontrado = productos.find { it.id == id }
        if (id.toInt() > productos.size || id.toInt() < 1) {
            println("El producto que buscas no existe")
            return false
        }
        if (productoEncontrado != null) {
            return productoEncontrado
        }
    } catch (ex: ClassCastException) {
        println("El producto que buscas no existe")
        return false
    } catch (ex: NumberFormatException) {
        println("Seleccionar una opcion válida  por favor")
        return false
    }
    return false
}
```

```
override fun listUsers() {
    activeUsers.forEach { println(it.name) }
}
```

```
private fun isValidUser(user: String, userPassword: String): Boolean {
    val user = activeUsers.find { it.name == user }
    if (user != null && user.password == userPassword) {
        return true
    }
    return false
}
```

Condiciones

Algunas de las condiciones utilizadas son las siguientes.

```
if (isValidUser(usuario, userPassword)) {
    println()
    println("********************** Bienvenido $usuario ******************************")
    compra.inicioCompras()
} else
    println("usuario/contraseña  incorrecto..intenta nuevamente.")
```

```
if (user != null && user.password == userPassword) {
    return true
}
```

```
when (opcion) {
    1 -> {
        println("Registrar usuario nuevo")
        user.altaUser()
    }

    2 -> {
        println(" Iniciar Sesión")
        user.login()
    }

    3 -> {
        println("Descargando usuarios.....")
        GlobalScope.launch { this: CoroutineScope
            delay( timeMillis: 1000)
            user.listUsers()
        }
        Thread.sleep( millis: 2000)

    }

    4 -> {
```

Bucles

También se utilizaron bucles para regenerar los procesos terminados, con excepciones o con incumplimientos.

```
do {
    println()
    println("SELECCIONAR  PRODUCTOS...")
    // se imprime la lista de productos
    producto.listProducts()
    println("s .-  salir")
    // seleccionar producto

} while (!opcProd.equals("S"))
```

```
do {
    println()
    println("**********************************")
    println(" Bienvenido   *** SuperTianguis ***")
    println("**********************************")
    println(" MENU ")
    println(" 1.- Registrar Usuario ")
    println(" 2.- Iniciar Sesión")
    println(" 3.- listar Usuarios")
    println(" 4.- Salir de aplicación")
    try {
```

Estructuras de datos

Tammbién se utilizaron listas para la simulación de las bases de datos u otras persistencias de datos.

```
var listOfProduct = mutableListOf<Producto>()
```

```
var productos = mutableListOf<Producto>()
```

```
var activeUsers = mutableListOf<UserManager>()
```

Herencia

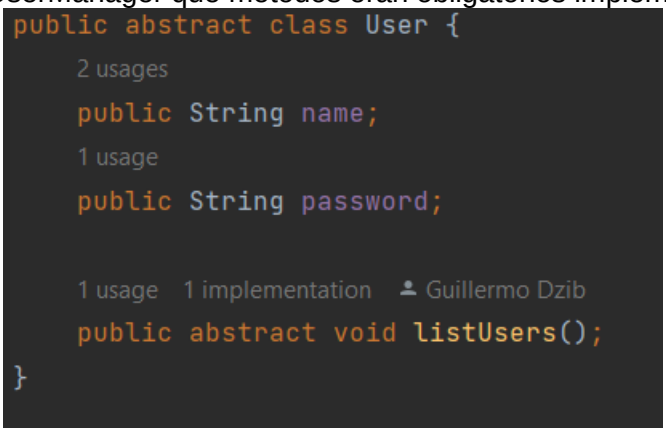
Se implementó que la clase UserManager sea clase hija de la clase User.

```
public abstract class User {
```

```
class UserManager( var name: String,  var password: String) : User() {
```

Clases abstractas

Se hizo a la clase User una clase abstracta, ya que su única función era decirle a la clase UserManager qué métodos eran obligatorios implementar.

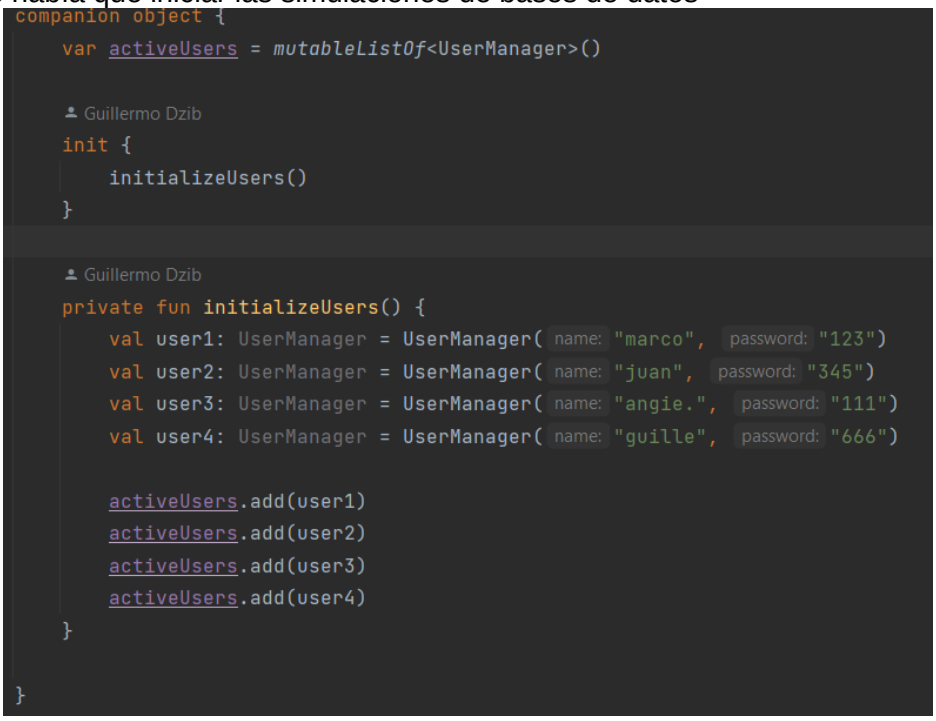


Data class

Se hizo una data class de los productos.

Companion Objects

Se utilizaron Companion objects en las clases que lo requerían. Como las clases en los que había que iniciar las simulaciones de bases de datos



Expresiones lambda

Se utilizaron funciones lambda.

```kotlin
val listProducts = {
    productos.forEach { it: Producto
        println("SKU: ${it.id}.- Producto: ${it.name} -> Precio: ${it.price} ")
    }
}

val totalProducts = { products: List<Producto>, cantidad: Int ->
    products.forEach { it: Producto
        val total = String.format("%.2f", it.price * cantidad)
        println("Producto: ${it.name} Precio: ${it.price} Total: ${total}")
    }
}
```

Interoperabilidad Java/Kotlin

Se hizo una clase en java, para comprobar la interoperabilidad.

```java
package clases;

1 usage  1 inheritor  Guillermo Dzib
public abstract class User {
    2 usages
    public String name;
    1 usage
    public String password;

    1 usage  1 implementation  Guillermo Dzib
    public abstract void listUsers();
}
```

Null safety

También se utilizaron todas las medidas de seguridad en donde eran necesarias para las llamadas seguras a métodos o validaciones de nulablidad.

```kotlin
val opcion = readLine()?.toInt() as Int
```

```
var opcProd: String = readLine()!!.toString().uppercase()
```

```
if (productoEncontrado != null) {
    return productoEncontrado
}
```

Manejo de excepciones

Casi para cualquier entrada de datos, por lo menos es necesario utilizar manejo de excepciones, en este caso para verificar que sean compatibles los casteos.

```
} catch (ex: InputMismatchException) {
    println("Enter valid number")
} catch (ex: NumberFormatException) {
    println("Seleccionar una opcion válida  por favor")
}
```

```
} catch (ex: NumberFormatException) {
    print("Digite una cantidad numérica porfavor")
    continue
}
```

Corrutinas

Se hizo una simulación de una descarga de datos para aplicar las corrutinas.

```
println("Descargando usuarios.....")
GlobalScope.launch { this: CoroutineScope
    delay( timeMillis: 1000)
    user.listUsers()
}
Thread.sleep( millis: 2000)
```

CONCLUSIONES (DIFICULTADES PRESENTADAS, APRENDIZAJES)

Es muy útil realizar un ejercicio como este para el reforzamiento de lo aprendido en el curso, porque te hace repasar cada tema visto para aplicarlo. Además se presentan problemas que al solucionarlos te aportan experiencia y conocimientos extra. Los problemas más relevantes presentados durante la elaboración del programa son los siguientes.

- El proyecto no inició con Gradle, así que se agregaron librerías manualmente descargando archivos .jar de páginas oficiales.

- Se pretendía implementar una base de datos, pero las librerías como Room, que facilitan el trabajo, solo están disponibles para Android.

- Hubo dificultad para identificar el momento adecuado para iniciar y cerrar el bloque try para manejar excepciones.

- Fue complicado identificar qué clases generar para la lógica de la aplicación.