

Actividad | 2 | Programa 2 (parte 1)

Desarrollo de Aplicaciones Móviles IV

Ingeniería en Desarrollo de Software



academi**ag**lobal

TUTOR: Marco Rodríguez Tapia

ALUMNO: Carlos Fco Estrada Salazar

FECHA: 31/Jul/2025

ÍNDICE

INTRODUCCIÓN	3
DESCRIPCÓN	4
JUSTIFICACIÓN	5
DESARROLLO	
Codificación	6
Prueba del programa	13
CONCLUSIÓN	14

GitHub Link:

INTRODUCCIÓN

En el contexto actual, los servicios financieros digitales se han convertido en una herramienta esencial para la gestión eficiente y segura de las finanzas personales. Con la creciente necesidad de automatizar procesos bancarios y ofrecer a los usuarios mayor comodidad y accesibilidad, el desarrollo de aplicaciones bancarias en línea ha tomado un papel fundamental. En esta actividad, se presenta la primera parte del desarrollo de un programa para el Banco Mexicano, diseñado en el lenguaje de programación Swift, que simula las operaciones básicas de una banca en línea: depósito y retiro de fondos. Este programa tiene como objetivo brindar una experiencia interactiva y sencilla para que los usuarios puedan realizar transacciones básicas de manera segura.

El sistema se desarrollará en forma de consola, mostrando un menú con las opciones de depósito, retiro, consulta de saldo y salida. En esta primera etapa, se implementarán exclusivamente las funciones de **depósito** y **retiro**, con las validaciones necesarias para garantizar que las operaciones sean correctas. Por ejemplo, si un cliente intenta retirar fondos sin tener saldo, el sistema lo informará adecuadamente; si el usuario desea hacer múltiples depósitos o retiros, también podrá hacerlo en una misma sesión. La lógica del programa incluye preguntas adicionales para continuar con nuevas operaciones o salir del sistema. En conjunto, esta actividad representa un paso inicial en la construcción de soluciones bancarias digitales que se ajusten a las necesidades actuales de los clientes y al desarrollo de habilidades de programación orientadas a la práctica profesional.

DESCRIPCIÓN

La actividad planteada gira en torno al desarrollo de un programa de simulación para un sistema de banca en línea del “Banco Mexicano”, utilizando el lenguaje de programación Swift. Este contexto refleja una situación realista y actual, ya que en la era digital los bancos requieren herramientas automatizadas y accesibles que permitan a los usuarios administrar sus finanzas de manera remota, eficiente y segura. El objetivo principal de esta etapa del proyecto es implementar un menú interactivo que permita al cliente realizar dos funciones clave: **depósito** y **retiro**, con la posibilidad de repetir operaciones o realizar otras diferentes, brindando una experiencia flexible.

En esta primera parte, se solicita que el programa pueda recibir datos desde el teclado, como la cantidad a depositar o retirar, validando siempre la información ingresada. Por ejemplo, si se realiza un retiro por primera vez y no hay saldo disponible, el programa debe notificarlo al usuario. En los retiros posteriores, si ya se han hecho depósitos y existe saldo suficiente, el sistema debe verificar que la cantidad solicitada esté disponible antes de proceder. En caso contrario, debe alertar que no hay fondos suficientes.

Lo solicitado también busca fomentar el uso de estructuras de control como bucles, condicionales y funciones, así como la lógica para validar entradas del usuario. Todo esto no solo fortalece las habilidades técnicas del estudiante en Swift, sino que además desarrolla competencias importantes para la creación de aplicaciones bancarias seguras y eficientes. Esta práctica representa una introducción al desarrollo de sistemas financieros con flujos interactivos y control de datos sensibles como el saldo del cliente.

JUSTIFICACIÓN

El desarrollo de una solución en Swift para simular un sistema de banca en línea, como el que se plantea en esta actividad, resulta fundamental tanto desde el punto de vista técnico como educativo. Actualmente, los servicios bancarios digitales requieren plataformas eficientes, intuitivas y seguras que permitan a los usuarios realizar operaciones financieras básicas como depósitos, retiros, consultas de saldo o cierre de sesión sin la necesidad de acudir a una sucursal física. Este tipo de soluciones no solo mejora la accesibilidad y comodidad para los clientes, sino que también representa una práctica realista y útil para los desarrolladores en formación.

Al implementar esta aplicación en el lenguaje Swift, se aprovechan las ventajas de un entorno moderno y robusto, ampliamente utilizado en el desarrollo de aplicaciones para sistemas Apple. Además, el ejercicio permite al estudiante aplicar conceptos fundamentales de programación como estructuras de control, entradas por teclado, validación de datos, y creación de menús interactivos. Todo esto contribuye a fortalecer las competencias necesarias para el desarrollo de software orientado a la resolución de problemas del mundo real.

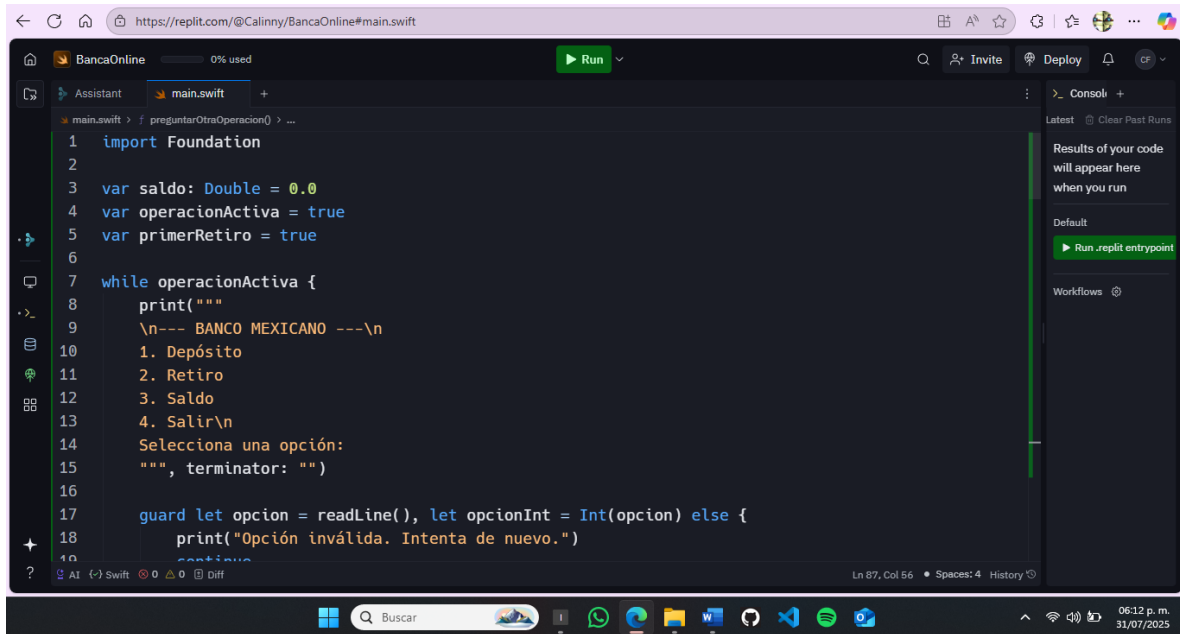
Este tipo de solución también es altamente escalable, ya que puede expandirse fácilmente para incluir funciones adicionales como transferencia entre cuentas, pagos de servicios o generación de estados de cuenta. Por ello, implementar esta solución no solo cumple con los requerimientos académicos de la actividad, sino que también prepara al estudiante para enfrentar desafíos reales en el desarrollo de aplicaciones financieras y comerciales.

DESARROLLO

Codificación

- **Paso 1: Crear la estructura principal del programa**

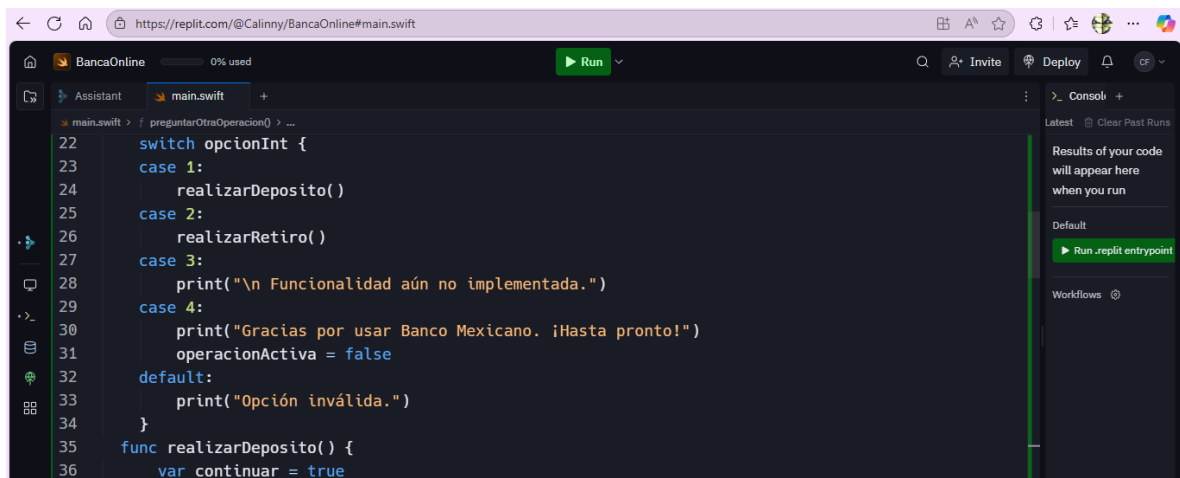
Utilizo un bucle “*while*” para mantener el menú activo hasta que el usuario decida salir. También, creo una variable global para almacenar el saldo.



```

1 import Foundation
2
3 var saldo: Double = 0.0
4 var operacionActiva = true
5 var primerRetiro = true
6
7 while operacionActiva {
8     print("""
9     \n--- BANCO MEXICANO ---\n
10    1. Depósito
11    2. Retiro
12    3. Saldo
13    4. Salir\n
14    Selecciona una opción:
15    """, terminator: "")
16
17     guard let opcion = readLine(), let opcionInt = Int(opcion) else {
18         print("Opción inválida. Intenta de nuevo.")
19     }
20     continue
21 }
  
```

- **saldo:** almacena el dinero actual del cliente.
- **operacionActiva:** controla si el usuario quiere seguir operando.
- **primerRetiro:** detecta si es la primera vez que intenta retirar dinero.



```

22 switch opcionInt {
23     case 1:
24         realizarDeposito()
25     case 2:
26         realizarRetiro()
27     case 3:
28         print("\n Funcionalidad aún no implementada.")
29     case 4:
30         print("Gracias por usar Banco Mexicano. ¡Hasta pronto!")
31         operacionActiva = false
32     default:
33         print("Opción inválida.")
34 }
35 func realizarDeposito() {
36     var continuar = true
  
```

• Paso 2: Función para depósito

```

35 func realizarDeposito() {
36     var continuar = true
37
38     while continuar {
39         print("\n¿Cuánto deseas depositar: ", terminator:"")
40         if let entrada = readLine(), let cantidad = Double(entrada), cantidad > 0 {
41             saldo += cantidad
42             print("\nDepósito exitoso. Nuevo saldo: ${saldo}")
43         } else {
44             print("\nCantidad inválida.")
45         }
46
47         print("\nDeseas realizar otro depósito (Sí/No): ", terminator:"")
48         if let respuesta = readLine()?.lowercased(), respuesta != "sí" && respuesta != "si" {
49             continuar = false
50         }
51     }
52 }
  
```

`var continuar = true`

- Se declara una variable booleana llamada “**continuar**” que controla el ciclo *while*.
- Mientras sea *true*, el usuario podrá seguir haciendo depósitos.

`while continuar {`

- Se inicia un bucle *while* que se ejecuta mientras el usuario quiera seguir depositando.
- Se muestra en consola un mensaje solicitando al usuario que escriba la cantidad a depositar.

`if let entrada = readLine(), let cantidad = Double(entrada), cantidad > 0 {`

Aquí se capturan los datos del teclado:

- *readLine()* lee lo que el usuario escribe.
- Se intenta convertir esa entrada a un número decimal (*Double*).
- Se valida que la cantidad sea mayor que cero.

```
saldo += cantidad
```

```
print("Depósito exitoso. Nuevo saldo: $(saldo)")
```

- Si la conversión y validación son exitosas, se suma la cantidad ingresada al saldo total.
- Luego se imprime el nuevo saldo actualizado.

```
} else {
```

```
    print("Cantidad inválida.")
```

```
}
```

- Si la conversión a número falla o la cantidad es negativa/cero, se muestra un mensaje de error.

```
if let respuesta = readLine()?.lowercased(), respuesta != "sí" && respuesta != "si" {
```

```
    continuar = false
```

```
}
```

- Se lee la respuesta del usuario y se convierte a minúsculas para evitar errores por mayúsculas.
- Si la respuesta no es “sí” ni “si”, el bucle termina cambiando continuar a false.

```
preguntarOtraOperacion()
```

- Una vez terminado el bucle de depósitos, se llama a otra función (preguntarOtraOperacion()) que probablemente pregunte al usuario si desea hacer otra acción (como retirar, consultar saldo o salir).

• Paso 3: Función para retiro

```

55 func realizarRetiro() {
56     if primerRetiro && saldo == 0 {
57         print("\nNo cuenta con saldo disponible.")
58         primerRetiro = false
59         preguntarOtraOperacion()
60         return
61     }
62
63     var continuar = true
64
65     while continuar {
66         print("\nCuánto deseas retirar: ", terminator:"")
67         if let entrada = readLine(), let cantidad = Double(entrada), cantidad > 0 {
68             if cantidad <= saldo {
69                 saldo -= cantidad
70                 print("\nRetiro exitoso. Saldo restante: ${saldo}")
71             } else {
72                 print("\nSaldo insuficiente. Saldo actual: ${saldo}")
73             }
74         }
75     }
76 }

```

```
if primerRetiro && saldo == 0 {
```

```
    print("No cuenta con saldo disponible.")
```

```
    primerRetiro = false
```

```
    preguntarOtraOperacion()
```

```
    return
```

```
}
```

- **Objetivo:** manejar la primera vez que el usuario intenta retirar dinero sin tener saldo.
- *primerRetiro* es una variable (fuera de esta función) que indica si es el primer intento.
- Si no hay saldo (*saldo == 0*) y es la primera vez (*primerRetiro == true*), se muestra un mensaje de advertencia.
- Luego se establece *primerRetiro = false* para que este mensaje no se repita en futuros intentos.
- Se llama a la función *preguntarOtraOperacion()* para saber si el usuario desea hacer otra acción.
- El *return* finaliza la función para evitar que el resto del código se ejecute.

```
var continuar = true
```

- Se declara una variable continuar que controla el ciclo del retiro.
- Mientras sea true, el usuario podrá hacer varios retiros seguidos.

```
while continuar {
```

- Se inicia un bucle que permite al usuario hacer múltiples retiros si así lo desea.

```
print("¿Cuánto deseas retirar?")
```

- Se solicita al usuario que ingrese la cantidad que desea retirar.

```
if let entrada = readLine(), let cantidad = Double(entrada), cantidad > 0 {
```

- Se lee la entrada del usuario desde el teclado.
- Se intenta convertir la entrada a un número decimal (*Double*).
- Se valida que la cantidad sea mayor que cero.

```
if cantidad <= saldo {
```

```
    saldo -= cantidad
```

```
    print("Retiro exitoso. Saldo restante: $\(saldo)")
```

```
}
```

Si la cantidad ingresada es menor o igual al saldo disponible:

- Se descuenta del saldo.
- Se imprime el nuevo saldo.

```
else {
    print("Saldo insuficiente. Saldo actual: $\"(saldo)\")
}
```

- Si el usuario intenta retirar más dinero del que tiene disponible, se muestra un mensaje de saldo insuficiente.

```
} else {
    print("Cantidad inválida.")
}
```

- Si la entrada no es válida (texto, número negativo, vacío, etc.), se muestra un mensaje de error.

```
print("¿Deseas realizar otro retiro? (Sí/No)")
```

- Se le pregunta al usuario si desea hacer otro retiro.

```
if let respuesta = readLine()?.lowercased(), respuesta != "sí" && respuesta != "si" {
    continuar = false
}
```

- Se lee la respuesta del usuario y se convierte a minúsculas para asegurar que se procese correctamente.
- Si el usuario responde algo distinto de “sí” o “si”, el bucle se detiene (continuar = false).

```
preguntarOtraOperacion()
```

- Al finalizar todos los retiros (o si el usuario ya no quiere continuar), se llama a otra función que pregunta si desea hacer otra operación como depósito, consultar saldo o salir.

- **Paso 5: Función para preguntar si desea hacer otra operación**

Esta función se llama después de cada depósito o retiro.

```

81     }
82 }
83
84 preguntarOtraOperacion()
85 }
86 func preguntarOtraOperacion() {
87     print("\nDeseas realizar otra operación (Sí/No): ", terminator:"")
88     if let respuesta = readLine()?.lowercased(), respuesta != "s" && respuesta != "si" {
89         operacionActiva = false
90         print("\nGracias por usar Banco Mexicano. ¡Hasta pronto!")
91     }
92 }
93
94
95 }
96

```

`print("\nDeseas realizar otra operación (Sí/No): ", terminator:"")`

- Muestra en consola la pregunta: "¿Deseas realizar otra operación (Sí/No)?".
- `\n` sirve para agregar un salto de línea antes del mensaje.
- El parámetro `terminator:""` hace que el cursor se quede en la misma línea después del mensaje (en lugar de irse a la siguiente línea).

`operacionActiva = false`

- Se asigna el valor `false` a la variable global `operacionActiva`.
- Al ponerla en `false`, se detiene el programa principal, es decir, el usuario ya no volverá al menú de opciones.

`print("\nGracias por usar Banco Mexicano. ¡Hasta pronto!")`

- Se muestra un mensaje de despedida para cerrar la sesión del usuario con una buena experiencia.

Prueba del programa

```

> Console
+
Show Only Latest Clear Past Runs
swift main.s... Ask Assistant
--- BANCO MEXICANO ---
1. Depósito
2. Retiro
3. Saldo
4. Salir
Selecciona una opción: 2
No cuenta con saldo disponible.
Deseas realizar otra operación (Sí/No): 

```

```

> Console
+
Show Only Latest Clear Past Runs
swift main.s... Ask Assistant
1. Depósito
2. Retiro
3. Saldo
4. Salir
Selecciona una opción: 1
¿Cuánto deseas depositar: 15875
Depósito exitoso. Nuevo saldo: $15875.0
Deseas realizar otro depósito (Sí/No): si
¿Cuánto deseas depositar: 125
Depósito exitoso. Nuevo saldo: $16000.0
Deseas realizar otro depósito (Sí/No): no
Deseas realizar otra operación (Sí/No): si
--- BANCO MEXICANO ---

```

Se muestra evidencia de lo solicitado en la actividad, **Deposito**; si es la primera vez que ingresa el usuario y desea realizar un retiro, el sistema indica el error de que no cuenta con saldo para retirar. Así como los mensajes y operaciones solicitados.

```

> Console
+
Show Only Latest Clear Past Runs
swift main.s... Ask Assistant
1. Depósito
2. Retiro
3. Saldo
4. Salir
Selecciona una opción: 2
Cuánto deseas retirar: 5900
Retiro exitoso. Saldo restante: $10100.0
Deseas realizar otro retiro (Sí/No): SI
Cuánto deseas retirar: 50
Retiro exitoso. Saldo restante: $10050.0
Deseas realizar otro retiro (Sí/No): no
Deseas realizar otra operación (Sí/No): Si
--- BANCO MEXICANO ---

```

```

> Console
+
Show Only Latest Clear Past Runs
swift main.s... Ask Assistant
Deseas realizar otra operación (Sí/No): Si
--- BANCO MEXICANO ---
1. Depósito
2. Retiro
3. Saldo
4. Salir
Selecciona una opción: 3
Funcionalidad aún no implementada.
--- BANCO MEXICANO ---
1. Depósito
2. Retiro
3. Saldo
4. Salir
Selecciona una opción: 4
Gracias por usar Banco Mexicano. ¡Hasta pronto!

```

Retiro; se muestra evidencia de lo que realiza el usuario al ingresar a esta opción. Se muestran los mensajes y operaciones solicitadas. Al ingresar a la opción **Saldo**, el sistema muestra el error, *funcionalidad aún no implementada*. **Salir**; el sistema se despide y se cierra.

CONCLUSIÓN

La realización de esta actividad representa un acercamiento práctico a los fundamentos de la programación aplicada al desarrollo de soluciones para el sector financiero. Diseñar una aplicación de banca en línea en lenguaje Swift no solo fortalece habilidades técnicas como el control de flujo, manejo de variables, validación de entradas y uso de ciclos, sino que también permite comprender la lógica detrás de procesos comunes que todos usamos en la vida diaria, como lo son los depósitos y retiros.

Desde una perspectiva laboral, este tipo de prácticas resulta esencial para quienes se desarrollan en áreas como la ingeniería en desarrollo de software, ya que permite crear soluciones funcionales, seguras y comprensibles para el usuario final. También resalta la importancia de considerar los posibles errores o comportamientos del usuario, como introducir datos inválidos o intentar retirar más de lo disponible, lo cual debe manejarse cuidadosamente para evitar fallos o experiencias negativas.

En la vida cotidiana, entender cómo funcionan estas aplicaciones puede ayudarnos como usuarios a ser más conscientes del manejo de nuestra información financiera y confiar más en la tecnología. Además, tener la capacidad de desarrollar herramientas similares podría incluso representar una oportunidad de emprendimiento en el mundo digital actual, donde las soluciones personalizadas para pequeñas instituciones o negocios son cada vez más demandadas.