



**UNIVERSIDAD TECNOLÓGICA DE PANAMÁ**  
**FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES**

**Programación 1**

**Proyecto Semestral**

**Daniella De León 8-1011-1020**

**Víctor Villarreal 8-1016-1917**

**Manuel Campos 8-1022-1118**

**Luz De León 8-1020-247**

**Diego Santimateo 9-764-2382**

**RODRIGO YÁNGÜEZ**

**1SF122**

# Introducción

En el curso de Programación I, hemos tenido la oportunidad de adentrarnos en los fundamentos de la programación con Java. Desde el principio, hemos aprendido a manejar la sintaxis del lenguaje y a utilizar estructuras para controlar el flujo de nuestros programas. Sin embargo, lo que realmente ha marcado una diferencia en nuestra formación ha sido la programación orientada a objetos (POO). Este enfoque ha sido clave para nosotros, ya que nos ha permitido organizar y gestionar los datos de manera más intuitiva mediante la creación de clases y objetos.

Uno de los aspectos más enriquecedores del curso ha sido la aplicación práctica de estos conceptos en los laboratorios. Como culminación de nuestro aprendizaje, hemos desarrollado un sistema de finanzas personales. Este proyecto final no solo refleja nuestra comprensión de los conceptos teóricos, sino que también pone en práctica la programación orientada a objetos en un contexto real.

Nuestro sistema de finanzas personales llamado *“My Finance Pal”* está diseñado para proporcionar una herramienta intuitiva y completa que facilite a las personas naturales a gestionar, visualizar y controlar sus activos y pasivos de manera eficiente. El proyecto nos ha dado la oportunidad de integrar múltiples aspectos de la programación, desde el diseño en UML, la creación de objetos y clases para modelar aspectos financieros; como ingresos, gastos, cuentas bancarias, préstamos. hasta la creación de interfaces gráficas con *“Java Swing”*, el uso de excepciones, la interacción con bases de datos mediante *“MySQL”* e importación de librerías para el correcto funcionamiento como *“JDatePicker”*, *“Mariadb-java-client”* y *“JfreeChart”*, entre otros aspectos significativos.

En resumen, este proyecto de finanzas personales, además de poner en práctica los conceptos fundamentales aprendidos durante el curso, demuestra nuestra capacidad para desarrollar soluciones tecnológicas aplicables en el mundo real. Al integrar la programación orientada a objetos, el diseño de interfaces gráficas y la gestión de bases de datos, hemos creado una herramienta que no solo organiza y simplifica la gestión financiera, sino que también fomenta una planificación y un control económico más efectivos. Estamos entusiasmados con el potencial de este sistema para proporcionar a los usuarios una visión clara y completa de su situación financiera y confiamos en que nos servirá como una valiosa contribución a nuestro desarrollo académico y profesional.

# Tabla de contenidos

<b>Introducción.....</b>	<b>2</b>
Tabla de contenidos.....	3
<b>Objetivos y Alcance del Proyecto.....</b>	<b>4</b>
• Proporcionar una plataforma para la gestión financiera personal.....	4
• Facilitar el seguimiento y análisis de transacciones financieras.....	4
• Promover la planificación y el control financiero.....	4
• Ofrecer soporte para la toma de decisiones financieras informadas.....	4
• Mejorar la estabilidad financiera a largo plazo.....	4
<b>Diagrama UML.....</b>	<b>5</b>
Bosquejo General Del Diagrama UML.....	6
Diagrama UML por partes.....	6
Diagrama de Base de Datos Relacional.....	13
<b>Guía de usuario.....</b>	<b>14</b>
• Primera Pantalla.....	14
• Registrar Usuario.....	14
• Iniciar Sesión.....	14
• En Página principal.....	14
• Cuentas bancarias.....	15
• En movimiento de cuentas.....	15
• Plazos Fijos.....	15
• Ingresos y Gastos.....	15
• Stocks.....	16
• Tarjeta de Crédito.....	16
• Préstamo.....	16
<b>Funcionalidades y características:.....</b>	<b>17</b>
• Gestión de Ingresos y Gastos.....	17
• Intereses Acumulados.....	17
• Ingresos Repetitivos.....	17
• Análisis de Activos y Pasivos.....	17
• Gestión de Deudas y Préstamos.....	17
• Manejo de Tarjetas de Crédito.....	17
• Visualización y Reportes.....	17
<b>Referencias bibliográficas.....</b>	<b>18</b>

# Objetivos y Alcance del Proyecto

- **Proporcionar una plataforma para la gestión financiera personal:**

Crear un sistema que permita a los usuarios gestionar todos los aspectos de sus finanzas personales de manera centralizada y accesible.

- **Facilitar el seguimiento y análisis de transacciones financieras:**

Ofrecer herramientas que permitan a los usuarios registrar y analizar sus transacciones financieras diarias, ayudándoles a comprender mejor su flujo de efectivo.

- **Promover la planificación y el control financiero:**

Ayudar a los usuarios a establecer metas financieras y a realizar un seguimiento de su progreso hacia el logro de estas metas, fomentando una planificación financiera sólida.

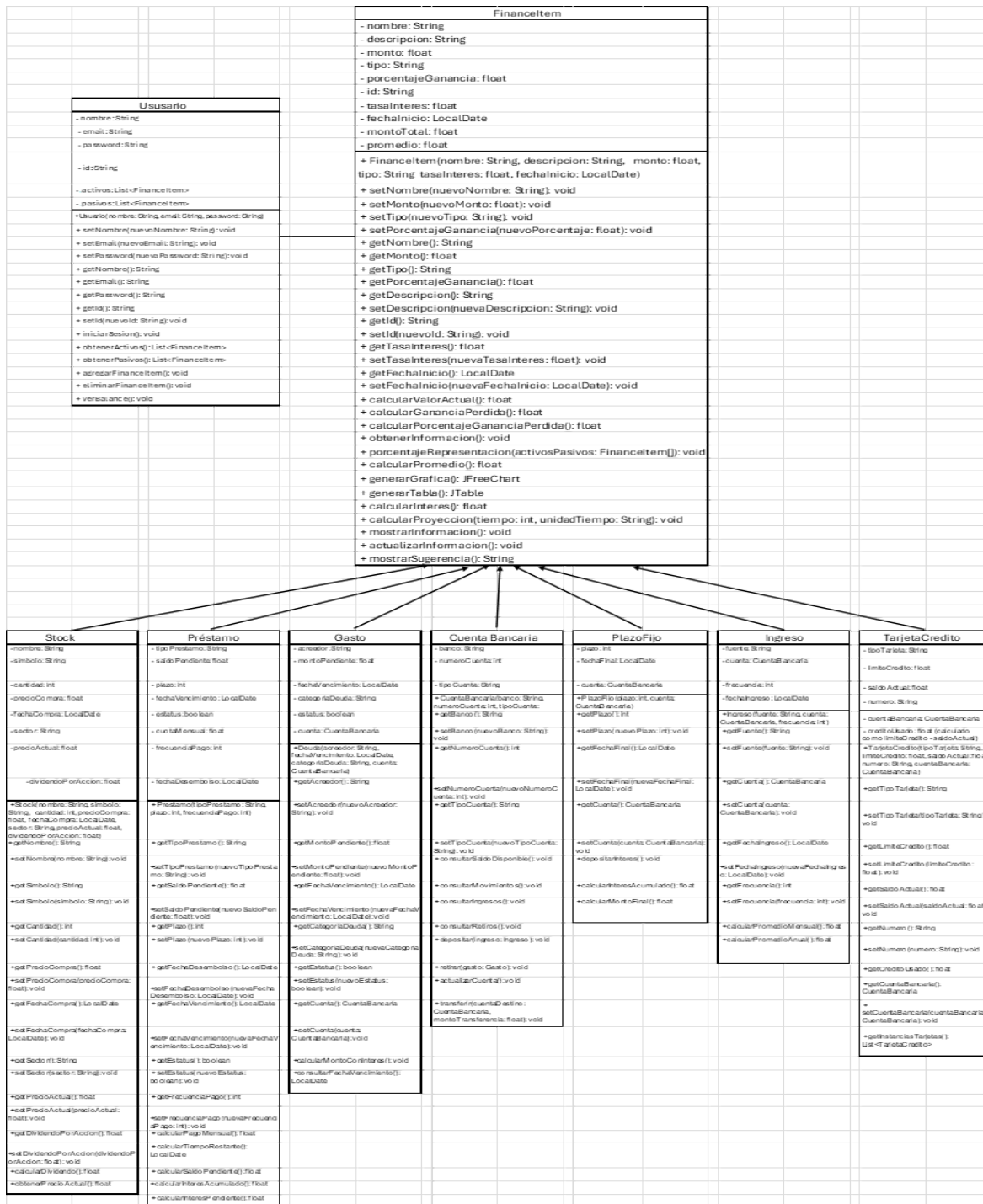
- **Ofrecer soporte para la toma de decisiones financieras informadas:**

Proporcionar datos y análisis que permitan a los usuarios tomar decisiones financieras más informadas y estratégicas.

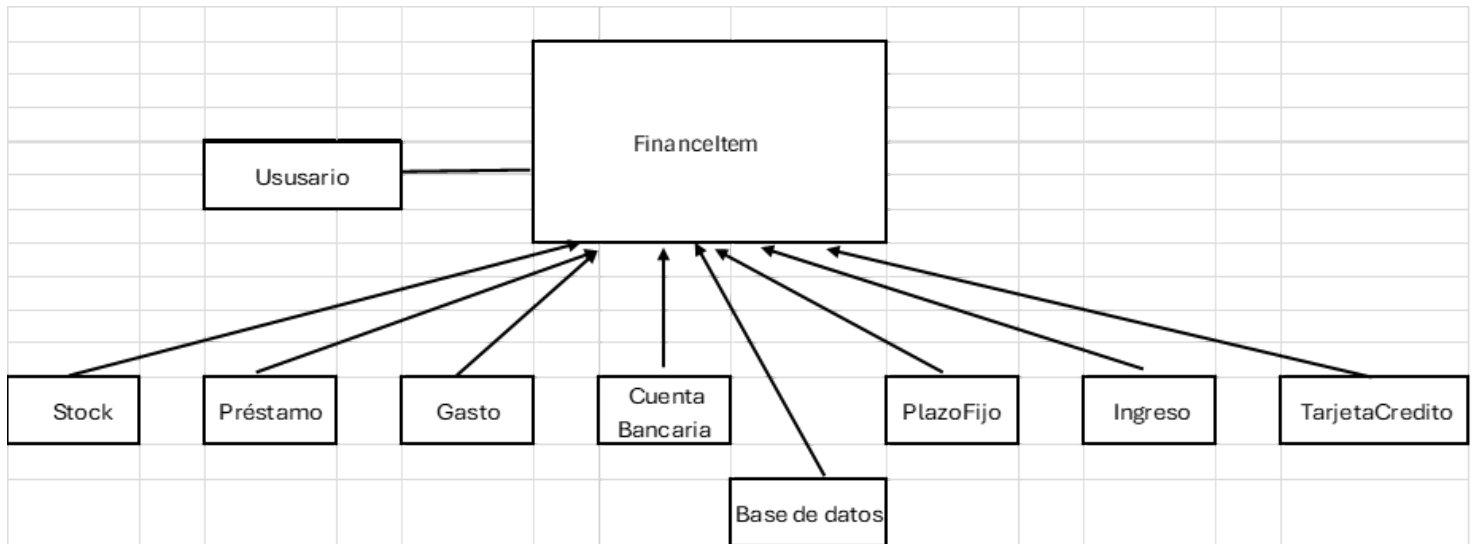
- **Mejorar la estabilidad financiera a largo plazo:**

Ayudar a los usuarios a gestionar sus recursos financieros de manera eficiente, promoviendo una mayor estabilidad económica y bienestar financiero a largo plazo.

# Diagrama UML



## Bosquejo General Del Diagrama UML



## Diagrama UML por partes

### Usuario

- nombre: String  
- email: String  
- contrasena: String  
- id: String  
- activos: ArrayList<FinanceItem>  
- pasivos: ArrayList<FinanceItem>

+ Usuario(nombre: String, email: String, contrasena: String)  
+ setNombre(nuevoNombre: String): void  
+ setEmail(nuevoEmail: String): void  
+ setContrasena(nuevaContrasena: String): void  
+ getNombre(): String  
+ getEmail(): String  
+ getContrasena(): String  
+ getId(): String  
+ setId(nuevoId: String): void  
+ getActivos(): ArrayList<FinanceItem>  
+ getPasivos(): ArrayList<FinanceItem>  
+ obtenerActivos(): List<FinanceItem>  
+ obtenerPasivos(): List<FinanceItem>  
+ agregarFinanceItem(item: FinanceItem): void  
+ eliminarFinanceItem(item: FinanceItem, esActivo: boolean): void  
+ verBalance(): void  
+ obtenerInformacionGeneral(): StringBuilder  
+ guardarClienteEnBaseDeDatos(): void  
+ cargarUsuariosDesdeBaseDeDatos(email: String): void  
+ correoExistente(email: String): boolean  
+ calcularBalance(): float  
+ getIdUsuarioActual(): String  
+ setUsuarioActual(usuario: Usuario): void  
+ getUsuarioActual(): Usuario

## FinanceItem

- nombre: String
- descripcion: String
- monto\_original: float
- tipo: String
- porcentaje\_ganancia: float
- ganancia\_perdida: float
- id: String
- tasa\_interes: float
- interes: float
- fecha\_inicio: LocalDate
- monto\_actual: float
- promedio\_mensual: float

- + FinanceItem(nombre: String, descripcion: String, monto\_original: float, tipo: String, tasa\_interes: float, fecha\_inicio: LocalDate)
- + FinanceItem(nombre: String, descripcion: String, montoOriginal: float, tipo: String, fechaInicio: LocalDate)
- + getNombre(): String
- + getDescripcion(): String
- + getMontoOriginal(): float
- + getTipo(): String
- + getPorcentajeGanancia(): float
- + getId(): String
- + getTasaInteres(): float
- + getInteres(): float
- + getFechaInicio(): LocalDate
- + getMontoActual(): float
- + getPromedioMensual(): float
- + getGanaciaPerdida(): float
- + setNombre(nombre: String): void
- + setDescripcion(descripcion: String): void
- + setMontoOriginal(monto\_original: float): void
- + setTipo(tipo: String): void
- + setPorcentajeGanancia(porcentaje\_ganancia: float): void
- + setId(id: String): void
- + setTasaInteres(tasa\_interes: float): void
- + setInteres(interres: float): void
- + setFechaInicio(fecha\_inicio: LocalDate): void
- + setMontoActual(monto\_actual: float): void
- + setPromedioMensual(promedioMensual: float): void
- + setGanaciaPerdida(ganancia\_perdida: float): void
- + calcularValorActual(): float
- + calcularGanaciaPerdida(): float
- + calcularPorcentajeGananciaPerdida(): float
- + obtenerInformacionGeneral(): StringBuilder
- + obtenerInformacionSubclase(): StringBuilder
- + obtenerInformacionCompleta(): void
- + calcularPorcentajeRepresentacion(activosPasivos: List<FinanceItem>): float
- + calcularPromedioMensual(): float
- + calcularPromedioAnual(): float
- + calcularInteres(): float

```
+ calcularProyeccion(tiempo: int, unidadTiempo: String): void
+ mostrarInformacion(): void
+ actualizarInformacion(): void
+ redonderCantidad(cantidad: float): float
```

#### Base de datos

```
- con: Connection
```

```
+ establecerConexion():void
+ realizarConsultaSelect(consulta:String,parametros:String[]): ResultSet
+ realizarConsultaSelectInterna(consulta:String):  ResultSet
+ ejecutarActualizacion(consulta:String, parámetros:String[]): boolean
+ mostrarDatosTabla(id_usuairo:String,tabla:String): void
+ cerrarConexion(): void
```

#### Stock

```
- nombre: String
- simbolo: String
- cantidad: int
- precioCompra: float
- fechaCompra: LocalDate
- sector: String
- precioActual: float
- dividendoPorAccion: float
- dividendoAcumulado: float
- dividendoEstimado: float
- frecuenciaDividendos: int
- cantidadInstancias: int
- instanciasStocks: List<Stock>
```

```
+ Stock(nombre: String, descripcion: String, fechaInicio: LocalDate, nombreEmpresa:
String, simbolo: String, cantidad: int, precioCompra: float, sector: String,
dividendoPorAccion: float, frecuenciaDividendos: int)
+ Stock(nombre: String, descripcion: String, fechaInicio: LocalDate, nombreEmpresa:
String, simbolo: String, cantidad: int, precioCompra: float, sector: String)
+ getNombreEmpresa(): String
+ getSimbolo(): String
+ getCantidad(): int
+ getPrecioCompra(): float
+ getSector(): String
+ getDividendoPorAccion(): float
+ getDividendoAcumulado(): float
+ getDividendoEstimado(): float
+ getCantidadInstancias(): int
+ getPrecioActual(): float
```



```
+ getFrecuenciaDividendos(): int
+ setNombreEmpresa(nombre_empresa: String): void
+ setSimbolo(simbolo: String): void
+ setCantidad(cantidad: int): void
+ setPrecioCompra(precio_compra: float): void
+ setSector(sector: String): void
+ setDividendoPorAccion(dividendo_por_accion: float): void
+ setDividendoAcumulado(dividendo_acumulado: float): void
+ setDividendoEstimado(dividendo_estimado: float): void
+ setPrecioActual(precio_actual: float): void
+ calcularDividendoAcumulado(): float
+ calcularDividendoFuturo(meses: int): float
+ obtenerPrecioActual(): float + calcularPorcentajeRepresentacionPorSimbolo(simbolo:
String): void
+ guardarStockBaseDatos(id_usuario: String): void
+ obtenerStocksBaseDatos(id_usuario: String): void
- extraerPrecioActual(jsonString: String): String
- obtenerPreciosMensuales(): List<Float>
- extraerPreciosMensuales(jsonString: String): List<Float>
- obtenerPreciosAnuales(): List<Float>
- extraerPreciosAnuales(jsonString: String): List<Float>
```

## Préstamo

- tipo\_prestamo: String
- saldo\_pendiente: float
- plazo: int
- fecha\_vencimiento: LocalDate
- estatus: int - cuota\_mensual: float
- cuenta\_bancaria: CuentaBancaria
- cantidad\_instancias: int
- instancias\_prestamos: List<Prestamo>

+ Prestamo(nombre: String, descripcion: String, monto\_original: float, tasa\_interes: float, fecha\_inicio: LocalDate, tipo\_prestamo: String, plazo: int, cuenta\_bancaria: CuentaBancaria)

- + getTipoPrestamo(): String
- + setTipoPrestamo(tipo\_prestamo: String): void
- + getSaldoPendiente(): float
- + setSaldoPendiente(saldo\_pendiente: float): void
- + getCuotaMensual(): float
- + setCuotaMensual(cuota\_mensual: float): void
- + getPlazo(): int
- + setPlazo(nuevo\_plazo: int): void
- + getFechaVencimiento(): LocalDate
- + setFechaVencimiento(fecha\_vencimiento: LocalDate): void
- + getEstatus(): int
- + setEstatus(nuevo\_estatus: int): void
- + getCuentaBancaria(): CuentaBancaria
- + calcularPagoMensual(): float + calcularTiempoRestante(): Period
- + calcularSaldoPendiente(): float
- + calcularInteresAcumulado(): float
- + calcularInteresTotal(): float
- + calcularInteresPendiente(): float
- + calcularPorcentajeRepresentacionPrestamo(): void
- + guardarPrestamoBaseDatos(): void
- + obtenerPrestamosBaseDatos(id\_usuario: String): void
- descontarCuota(): void
- calcularMontoPendiente(): float
- redonderCantidad(cantidad: float): float

## Gasto

- acreedor: String
- frecuencia: int
- sumatoria\_pagos: float
- categoria\_gasto: String
- estatus: boolean
- estatus\_entero: int
- cuenta: CuentaBancaria
- cantidad\_instancias: int (static)
- instancias\_gastos: List<Gasto> (static)

+ Gasto(nombre: String, descripcion: String, monto\_original: float, fecha\_inicio: LocalDate, acreedor: String, frecuencia: int, categoria\_gasto: String, cuenta: CuentaBancaria)

- + getAcreedor(): String
- + setAcreedor(nuevoAcreedor: String): void
- + getFrecuencia(): int
- + setFrecuencia(frecuencia: int): void
- + getSumatoriaPagos(): float
- + setSumatoriaPagos(nueva\_sumatoria\_pagos: float): void
- + getCategoriaGasto(): String
- + setCategoriaGasto(categoria\_gasto: String): void
- + getEstatus(): boolean
- + setEstatus(nuevoEstatus: boolean): void
- + getCuenta(): CuentaBancaria
- + setCuenta(cuenta: CuentaBancaria): void
- + getEstatus\_entero(): int
- + setEstatus\_entero(estatus\_entero: int): void
- + calcularValorActual(): float
- + obtenerInformacionSubclase(): StringBuilder
- + calcularPromedioMensual(): float
- + calcularPromedioAnual(): float
- + actualizarInformacion(): void
- + obtenerEstatusEntero(): int
- + guardarGastoBaseDatos(): void
- + calculaPorcentajeRepresentacionGasto(nombre: String): void
- + calcularPromedioMensualGasto(id\_usuario: String): float
- + calcularPromedioAnualIngresos(id\_usuario: String): float
- + obtenerGastosUltimosMeses(id\_usuario: String): List<Float>
- + obtenerGastosAnualesRecientes(id\_usuario: String): List<Float>

### Cuenta Bancaria

- banco: String
- numero\_cuenta: String
- tipo\_cuenta: String
- id\_usuario: String

- + CuentaBancaria(nombre: String, descripcion: String, monto\_original: float, tasaInteres: float, fecha\_inicio: LocalDate, banco: String, numero\_cuenta: String, tipo\_cuenta: String, id\_usuario: String)
- + getBanco(): String
- + getNumeroCuenta(): String
- + getTipoCuenta(): String
- + getIdUsuario(): String
- + setBanco(banco: String): void
- + setNumeroCuenta(numero\_cuenta: String): void
- + setTipoCuenta(tipo\_cuenta: String): void
- + setIdUsuario(id\_usuario: String): void
- + calcularValorActual(): float
- + obtenerInformacionSubclase(): StringBuilder
- + calcularPorcentajeRepresentacionSubclase(activosPasivos: List<FinanceItem>): float
- + calcularPromedioMensual(): float
- + calcularPromedioAnual(): float
- + actualizarInformacion(): void
- + registrarInteres(): void
- + guardarInteres(interés\_mensual: float, fecha\_deposito: LocalDate): void
- + calcularInteresSobreBalance(balance\_mensual: float): float
- + calcularBalanceMensual(fechaInicial: String, balance\_anterior: float): float
- + depositarMonto(monto: float): void
- + retirarMonto(monto: float): void
- + calcularBalanceActual(): float
- + calcularBalancePrevio(fecha\_final: String): float
- + calcularInteresAcumulado(): void
- + calcularBalanceAnual(fecha\_inicial: String, balance\_anterior: float): float
- + calcularBalancesMensualesRecientes(): List<Float>
- + calcularBalancesAnualesRecientes(): List<Float>
- + calcularPorcentajeRepresentacionCuenta(): void
- + guardarCuentaBancariaBaseDatos(): void
- + obtenerCuentasBancariasBaseDatos(id\_usuario: String): void

### Plazo Fijo

- plazo: int
- fecha\_final: LocalDate
- cuenta: CuentaBancaria
- cantidad\_instancias\_plazo\_fijo: int (static)
- +instancias\_plazos\_fijos: List<PlazoFijo> (static)

- +PlazoFijo(nombre: String, descripcion: String, montoOriginal: float, tasa\_interes: float, fecha\_inicio: LocalDate, plazo: int, cuenta: CuentaBancaria)
- +getPlazo(): int
- +setPlazo(nuevo\_plazo: int): void
- +getFechaFinal(): LocalDate

```

+setFechaFinal(nueva_fecha_final: LocalDate): void
+getCuenta(): CuentaBancaria
+setCuenta(cuenta: CuentaBancaria): void
+depositarInteres(): void
+calcularInteresAcumulado(): float
+calcularMontoFinal(): float
+guardarPlazoFijoEnBaseDatos(): void
+calcularValorActual(): float
+obtenerInformacionSubclase(): StringBuilder
+calcularPorcentajeRepresentacionSubclase(activosPasivos: List<FinanceItem>): float
+calcularPromedioMensual(): float
+calcularPromedioAnual(): float
+actualizarInformacion(): void
+encontrarPlazoFijoPorId(id: String): PlazoFijo
+calcularPorcentajeRepresentacionPlazoFijo(): void
+obtenerPlazoFijosBaseDatos(id_usuario: String): void
+eliminarPlazoFijoEnBaseDatos(): void

```

### Ingreso

```

- fuente: String
- cuenta_bancaria: CuentaBancaria
- frecuencia: int
+ cantidad_instancias: int (static)
+ instancias_ingresos: List<Ingreso> (static)

```

```

+ Ingreso(nombre: String, descripcion: String, monto_original: float, fecha_inicio:
LocalDate, Fuente: String, cuenta: CuentaBancaria, frecuencia: int)
+ getFuente(): String
+ getCuentaBancaria(): CuentaBancaria
+ getFrecuencia(): int
+ setFuente(String): void
+ setCuentaBancaria(CuentaBancaria): void
+ setFrecuencia(int): void
+ calcularValorActual(): float
+ obtenerInformacionSubclase(): StringBuilder
+ calcularPromedioMensual(): float
+ calcularPromedioAnual(): float
+ actualizarInformacion(): void
+ calcularPorcentajeRepresentacionIngreso(String): void
+ calcularPromedioMensualIngresos(String): float (static)
+ calcularPromedioAnualIngresos(String): float (static)
+ obtenerIngresosUltimosMeses(String): List<Float> (static)
+ obtenerIngresosAnualesRecientes(String): List<Float> (static)
+ guardarIngresoBaseDatos(): void
+ obtenerIngresosBaseDatos(String): void (static)

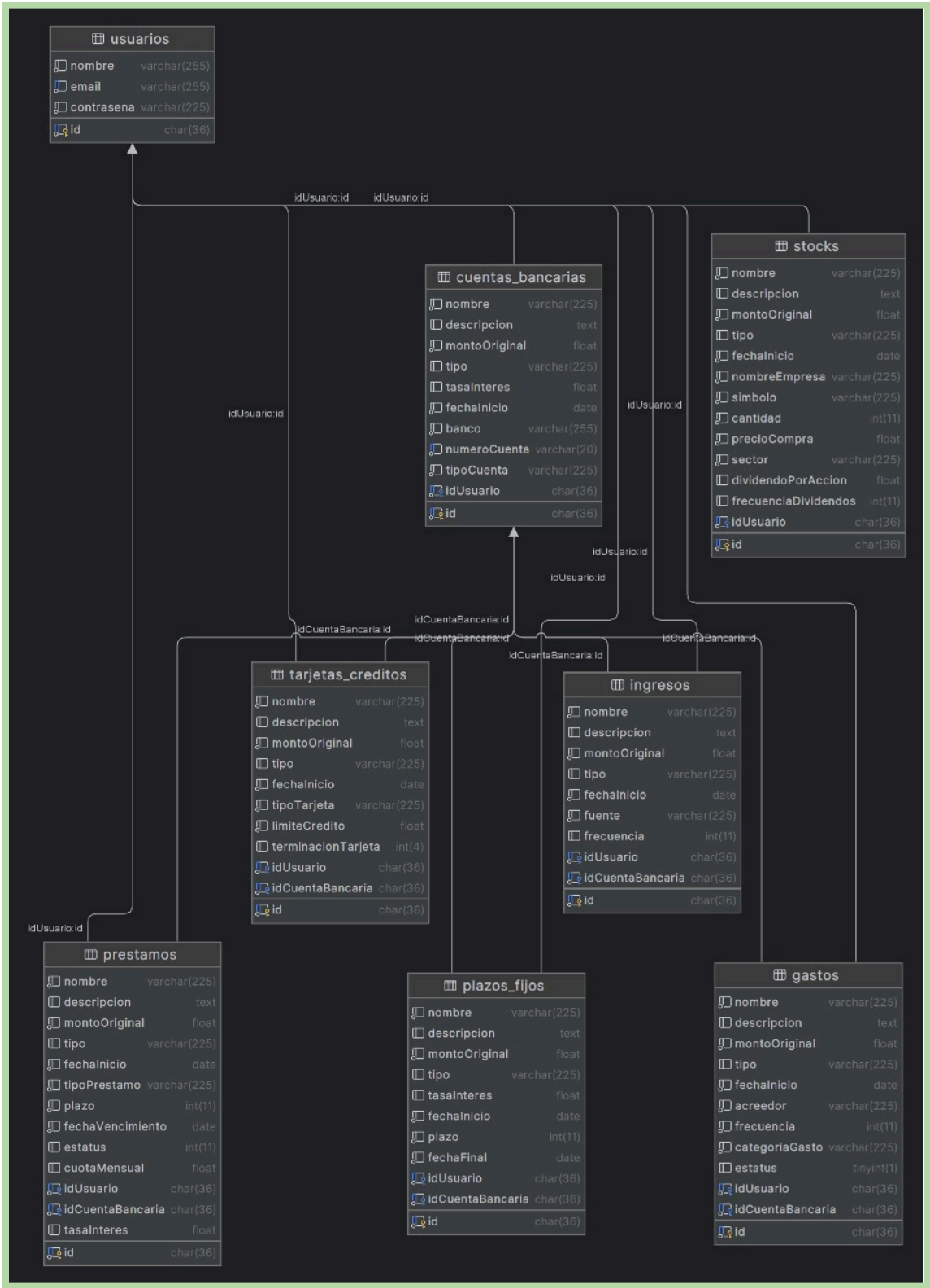
```

## TarjetaCredito

```
- tipoTarjeta: String
- limiteCredito: float
- saldoActual: float
- numero: int
- creditoUsado: float
- cuentaBancaria: CuentaBancaria
+ cantidad_instancias: int
+ instanciasTarjetas: List<TarjetaCredito> (static)
```

```
+ TarjetaCredito(nombre:String descripción: String, tipoTarjeta: String, fecha_inicio
limite_credito: float, monto_original: float, numero: int, cuentaBancaria:
cuenta_bancaria)
+ getTipoTarjeta(): String
+ setTipoTarjeta(tipo_tarjeta:String): void
+ getLimiteCredito(): float
+ setLimiteCredito(límite_credito:float): void
+ getSaldoActual(): float
+ setSaldoActual(saldo_actual:float): void
+ getNumero(): int
+ setNumero(numemero:int): void
+ getCreditoUsado(): float
+ getCuentaBancaria(): CuentaBancaria
+ setCuentaBancaria(CuentaBancaria): void
+ calcularPorcentajeRepresentacionTarjeta(): void
+ obtenerCantidadInstancias(): int (static)
+ obtenerInstancias(): List<TarjetaCredito> (static)
+ guardarTarjetaBaseDatos(): void
+ obtenerTarjetaCreditoBaseDatos(String): void (static)
+ pagarTarjeta(float): void
+ calcularValorActual(): float
+ obtenerInformacionSubclase(): StringBuilder
+ calcularPromedioMensual(): float
+ calcularPromedioAnual(): float
+ actualizarInformacion(): void
+ calcularPorcentajeRepresentacionSubclase(FinanceItem[]): float (static)
```

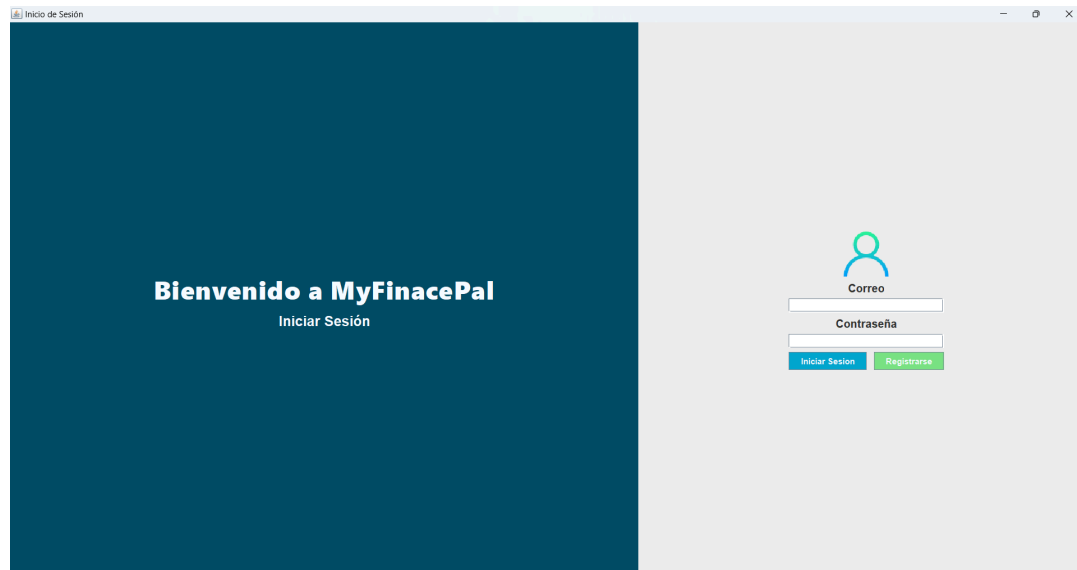
Diagrama de Base de Datos Relacional



# Guía de usuario

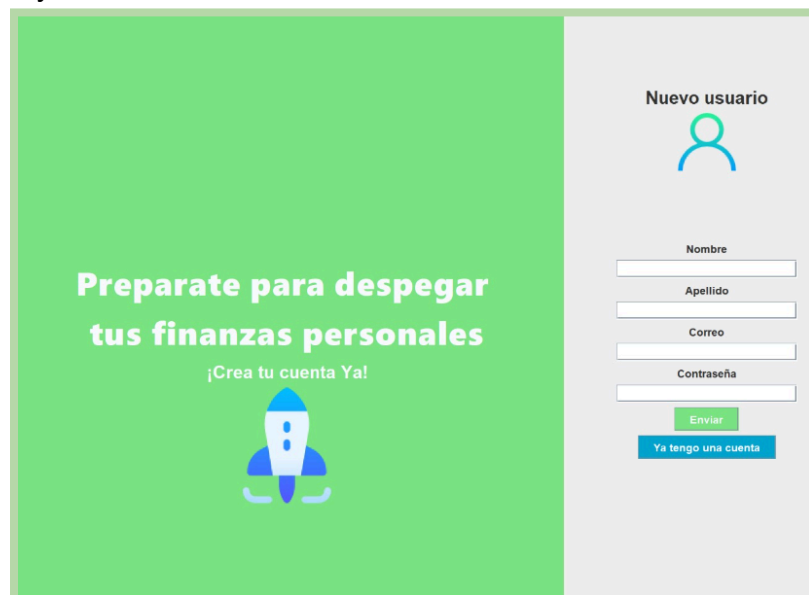
- **Primera Pantalla:**

Al abrir el software, se presentará la pantalla de Inicio de Sesión / Registro. En esta pantalla, el usuario deberá elegir entre las opciones: iniciar sesión o registrarse. Si es la primera vez que utiliza el software, deberá seleccionar la opción de “Registrarse” pulsando el botón correspondiente. Si se ha registrado previamente, deberá ingresar su correo electrónico y contraseña en los campos indicados.



- **Registrar Usuario:**

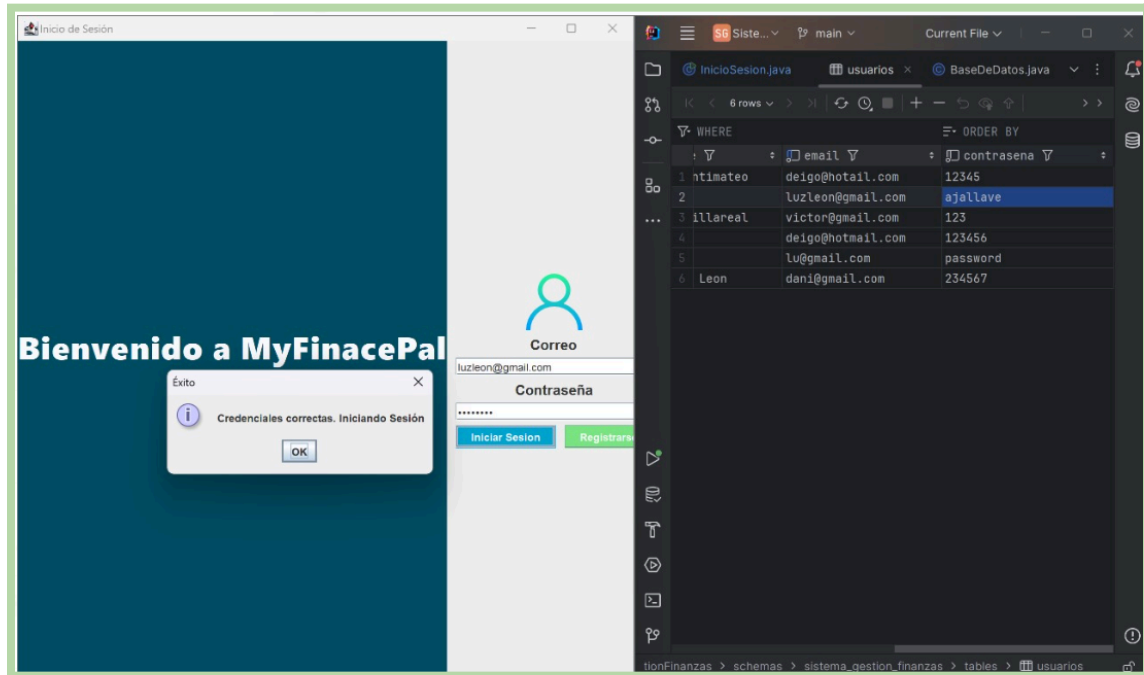
Si es la primera vez que el usuario accede a esta aplicación, deberá ingresar sus datos personales en la sección de *Registrarse*. Allí, se le pedirá que proporcione su nombre, apellido, correo electrónico y contraseña. Estos datos se guardan para que el usuario pueda iniciar sesión con su correo y contraseña.





- **Iniciar Sesión:**

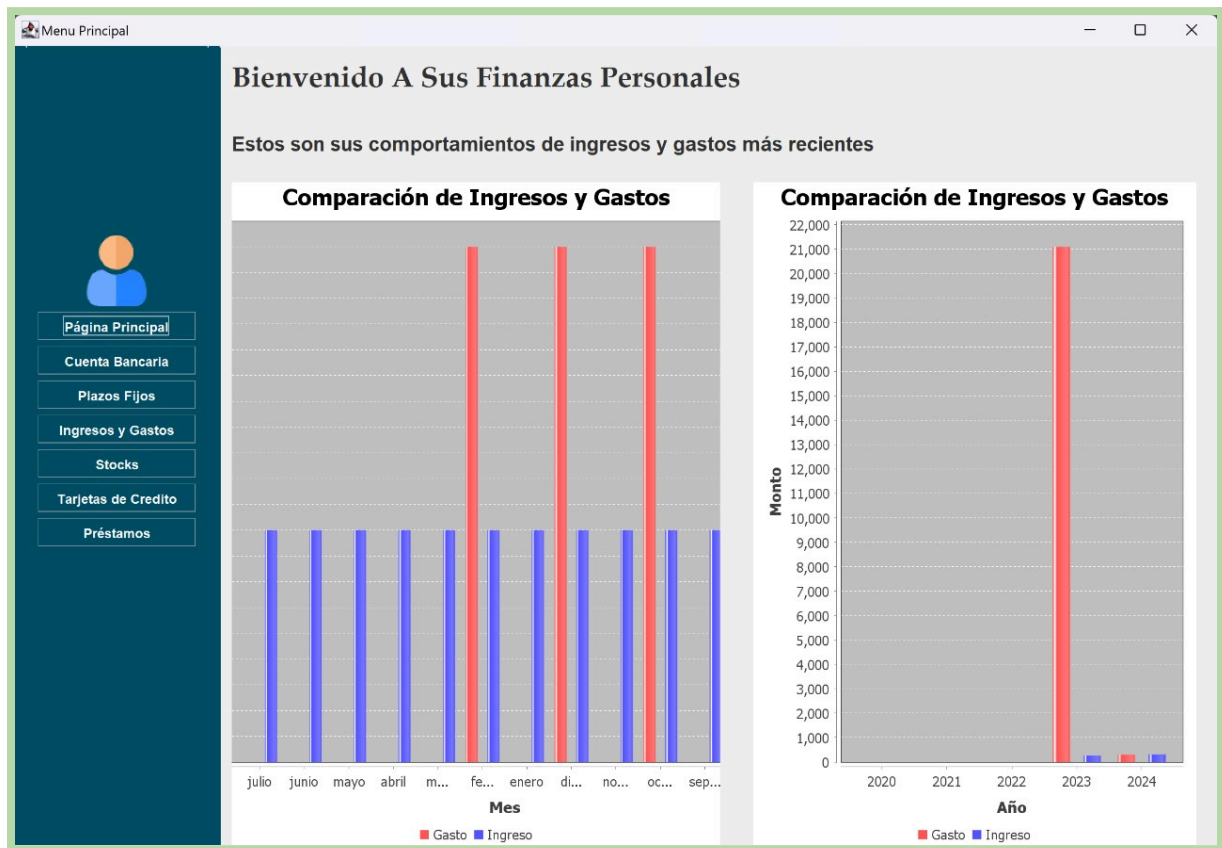
Si ya se ha registrado una cuenta previamente, el usuario deberá ingresar su dirección de correo electrónico y contraseña en los campos indicados y luego pulsar el botón "Iniciar Sesión". Una vez pulsado, se muestra un diálogo que le indica que su cuenta se ha iniciado correctamente de ser ese el caso, se le permitirá acceder a su cuenta y a los datos que ya se han registrado en el software. si no, se mandará un mensaje de error indicando que hay campos a corregir.



- Una vez iniciada la sesión en el menú principal, el usuario podrá acceder a las cuentas bancarias que posee, consultar los ingresos registrados, revisar los plazos fijos activos, administrar los stocks disponibles, ver el estado y detalles de las tarjetas de crédito, consultar la información de los préstamos vigentes y gestionar los gastos registrados.

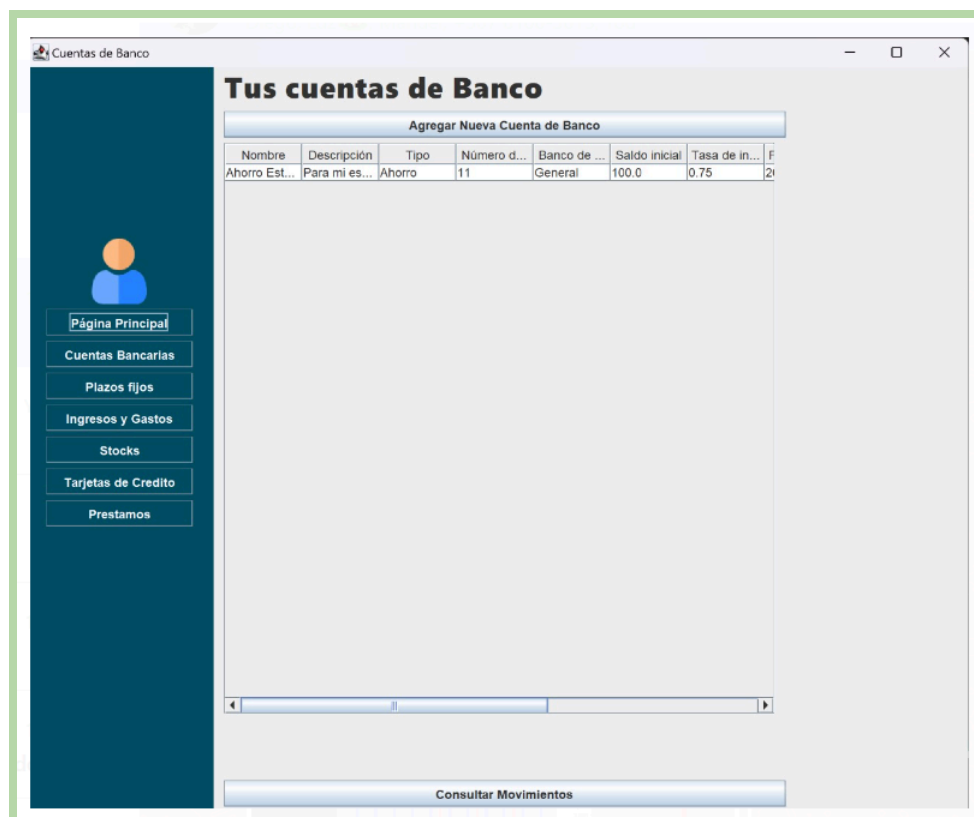
- **En Página principal:**

se encuentra la bienvenida a las finanzas personales del usuario, donde podrá observar de manera práctica y sencilla los ingresos y gastos más recientes de manera general, e incluyendo a gráficas que representan cada uno, aquí, el usuario podrá mirar de forma visual las ganancias y pérdidas que ha tenido en el último mes y en el último año.



- **Cuentas bancarias**

El usuario podrá observar todas sus cuentas de banco actuales. También, podrá agregar nuevas cuentas bancarias. Además, podrá consultar, en cualquier momento, los movimientos de una cuenta en específica.



- Para agregar una nueva cuenta bancaria, el usuario deberá pulsar el botón "*Agregar Nueva Cuenta de Banco*" que se encuentra en la parte superior del apartado cuentas bancarias. Luego deberá ingresar los datos necesarios como: nombre de la cuenta, descripción, tipo de cuenta, número de cuenta, banco de origen, saldo inicial y tasa de interés y fecha de creación de la cuenta. Una vez completados todos los campos solicitados, el usuario deberá pulsar el botón "*Crear*" para registrar la nueva cuenta bancaria, que estará disponible de inmediato para su consulta y gestión.

### Ingresa los datos para agregar una nueva cuenta

**Nombre de Cuenta**  
Nombre de ejemplo

**Descripción**  
Información extensa...

**Tipo de Cuenta**  
Selecciona una opción ▼

**Número de Cuenta**  
2342-3455-2333

**Banco de Origen**  
Banitsmo

**Saldo Inicial**  
123234.43

**Tasa Interés**  
2.3

**Fecha de Inicio**  
12/06/2024 ...

**Crear**

- **En movimiento de cuentas**


El usuario podrá revisar los movimientos de una cuenta bancaria específica. Para obtener esta información, deberá completar los campos solicitados: el número de cuenta, la fecha de los movimientos y el nombre de la cuenta. Una vez ingresados todos estos datos, el usuario deberá pulsar el botón "*Buscar*" y automáticamente la información deseada se visualizará en el recuadro.




The screenshot shows a web browser window with the title 'Consultar Movimiento'. The page has a dark blue background. At the top center, the title 'Movimiento de Cuentas' is displayed in white. Below this, there are two input fields: 'Número de Cuenta:' with the value '11' and 'Nombre de Cuenta:' with the value 'Ahorro Estudio'. To the right of these fields is a button labeled 'BUSCAR'. Below the input fields is a large white rectangular area, likely intended for displaying the search results.

- **Plazos Fijos**

El usuario podrá agregar o eliminar un plazo fijo. Para eliminar un plazo fijo ya creado, solamente deberá ingresar el ID del plazo fijo y pulsar el botón donde dice "*Eliminar Plazo Fijo*". En el apartado del lado derecho, el usuario podrá observar los tipos de intereses tales como: el interés actual, el interés mensual, el interés anual y el interés acumulado. Por otro lado, si el usuario desea agregar un Plazo Fijo, deberá llenar los datos que se le solicita tales como: el nombre, la descripción, el tipo, el monto original, la tasa de interés, la fecha de inicio, la fecha final, el plazo, y la cuenta de banco que el usuario desee. Una vez el usuario haya completado los datos que se le solicita, deberá pulsar a el botón que dice "Agregar Plazo Fijo".


Ingresos Y Gastos



Menu Principal

Cuentas Bancarias

Piazos Fijos

Ingresos y Gastos

Stocks

Tarjeta de credito

Prestamos

## Visualiza tus ingresos y gastos

Nombre

descripcion

Fuente

Cuenta de Banco

11 Ahorro Estudio

Frecuencia

12

Fecha

16/07/2024

Monto

134.32

Agregar Ingreso

Nombre

Acreedor

descripcion

Cuenta de Banco

11 Ahorro Estudio

Frecuencia

0

Fecha

22/07/2024

Monto

235.4

Categoria

Gastos Básicos

Agregar Gastos

ID	Nombre	Descripción	Fuente	Cuenta de banco
ca898...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
ca96b...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
ca9a1...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
ca9b0...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
ca9c6...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
ca9e6...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa08...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa20...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa30...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa43...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa53...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa68...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...
caa7f...	Trabajo	Mi trabajo en I...	Mi sudor y sangre	sistemagestion...

ID	Nombre	Acreedor	Descripción	Cuenta de banco
42ccd...	Comida	TacoBell	me lo merecia	sistemagestion...
69091...	Luz	Naturgy	Luz muy cara	sistemagestion...
6915e...	Luz	Naturgy	Luz muy cara	sistemagestion...
69171...	Luz	Naturgy	Luz muy cara	sistemagestion...
69181...	Luz	Naturgy	Luz muy cara	sistemagestion...

- Si el usuario desea agregar un gasto, deberá ir al lado derecho y colocar los datos que se requiere tales como: el nombre, el acreedor, la fuente, la cuenta de banco, la frecuencia, la fecha y el monto. Una vez que el usuario haya puesto todos los datos obligatorios, deberá pulsar al botón *“agregar gasto”*.

- **Stocks**

El usuario podrá conocer el estado de sus inversiones en acciones. Para ello, deberá primero agregar un nuevo stock. Para lograrlo, el usuario deberá colocar sus datos en los campos solicitados tales como: el nombre acción, el símbolo, la cantidad, la descripción, el precio de compra, el nombre empresa, el dividendo por acción, la frecuencia de dividendos. Una vez ya colocados los datos, el usuario deberá pulsar el botón *“agregar nuevo stock”* que se encuentra en la parte inferior. También, el usuario podrá observar en el recuadro ubicado en la parte inferior información general en la cual podrá obtener datos generales de los stocks correspondientes.

Nombre Acción	ID	Descripción	Nombre Empresa	Sector	Simbolo	Cantidad	Dividen
Apple	a9ea887d-4837-11ef-8519-b48c9d5d75f4	Mi primer stonk	Apple	Tecnología de la Información	AAPL	150	0.97

- **Tarjeta de Crédito**

El usuario podrá agregar una tarjeta a su cuenta bancaria. Para poder hacer este procedimiento, para esto, el usuario deberá ingresar la siguiente información: el tipo de tarjeta, el límite de crédito, el saldo actual, la terminación del número de tarjeta y la cuenta bancaria. Una vez que el usuario haya ingresado todos estos datos, deberá pulsar el botón “agregar” y, a su vez, podrá observar en la parte inferior un recuadro con la información ya antes colocada.

**Tus Tarjetas**

Tipo Tarjeta:

Limite Credito:

Saldo Actual:

Terminación del numero tarjeta:

Cuenta Bancaria:

	Tipo	Limite de Crédito	Saldo Actual	Número de Tarjeta	Cuenta Bancaria
Pasivo		14500.0	14500.0	6453	11 Ahorro Estudio

- **Préstamo**

El usuario podrá agregar un préstamo. Para poder realizarlo, el usuario deberá colocar la información que se le solicita tales como: el nombre, la descripción, la cuenta bancaria, el tipo, el monto, el plazo, la fecha de inicio, y el estatus. Una vez el usuario haya colocado todos los datos obligatorios, deberá presionar el botón “agregar Préstamo” para culminar dicho proceso.

**Tus Préstamos**

Nombre:

Descripción:

Cuenta Bancaria:

Tipo:

Estatus:

Fecha de Inicio:

Monto:

Plazo:

Taza de Interés:

ID	Nombre	Descripción	Monto Ori...	Fecha Inicio	Tipo Prést...	Plazo	Fecha Ve...	Cuota M...
----	--------	-------------	--------------	--------------	---------------	-------	-------------	------------

# Funcionalidades y características:

- **Gestión de Ingresos y Gastos**

El sistema permitirá a los usuarios registrar y controlar sus ingresos y gastos de manera detallada, ofreciendo una visión clara y precisa de su flujo de efectivo. Los usuarios podrán clasificar sus transacciones en diferentes categorías, facilitando el análisis y la planificación financiera.

- **Intereses Acumulados**

La funcionalidad para calcular y mostrar los intereses acumulados en diversas cuentas de ahorro y otros instrumentos financieros permitirá a los usuarios entender cuánto están ganando en intereses a lo largo del tiempo. Esto es crucial para la planificación a largo plazo y la maximización de los ahorros.

- **Ingresos Repetitivos**

El sistema identificará y registrará automáticamente los ingresos recurrentes, como salarios y pensiones. Una vez que estos ingresos sean ingresados y clasificados como recurrentes, el sistema los añadirá automáticamente en los períodos correspondientes, ahorrando tiempo y esfuerzo al usuario.

- **Análisis de Activos y Pasivos**

Proporcionará herramientas para analizar y comprender el porcentaje que representan los activos y pasivos dentro de la economía personal del usuario. Esto ayudará a los usuarios a obtener una visión más clara y detallada de su situación financiera, permitiéndoles tomar decisiones informadas y estratégicas.

- **Gestión de Deudas y Préstamos**

Los usuarios podrán llevar un registro detallado de todas sus deudas pendientes, incluyendo préstamos, hipotecas y otros compromisos financieros. Esta funcionalidad les ayudará a planificar y gestionar sus pagos de manera más eficiente, evitando el sobreendeudamiento y mejorando su estabilidad financiera.

- **Manejo de Tarjetas de Crédito**

Incluirá funcionalidades específicas para la administración de tarjetas de crédito, permitiendo a los usuarios seguir sus gastos y gestionar sus pagos pendientes. Esto les ayudará a evitar cargos por intereses adicionales y a mantener un buen historial crediticio.

- **Visualización y Reportes**

Ofrecerá opciones avanzadas de visualización y generación de reportes financieros. Los usuarios podrán generar reportes detallados que proporcionen una visión completa de su situación financiera, permitiéndoles identificar tendencias, detectar áreas de mejora y tomar decisiones informadas para optimizar su gestión financiera.



# Referencias bibliográficas

- *How do I implement JDatePicker.* (s. f.). Stack Overflow. <https://stackoverflow.com/questions/26794698/how-do-i-implement-jdatepicker>
- Repositorio: Dasantimate. (s. f.). *GitHub - dasantimate14/SistemaGestionFinanzasPersonales: El repositorio del proyecto semestral grupal para la materia de Programación I en la Universidad Tecnológica de Panamá. Es un proyecto que busca llevar un control de los activos y pasivos comunes de una persona natural. Incluye proyecciones aproximadas en base a los datos y gráficas para presentar resultados.* GitHub. <https://github.com/dasantimate14/SistemaGestionFinanzasPersonales>
- Plynko, P. (2023, 22 febrero). *Java JTable*. CodeGym. <https://codegym.cc/groups/posts/java-jtable>
- *Package javax.swing* (2020) *javax.swing (Java Platform SE 7 )*. Available at: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html> (Accessed: 22 July 2024).