Simulador de Banco
(Mayo de 2025)

Yenifer C. Contreras, Karen D. Martínez y Angie V.Martínez

Resumen – Este documento presenta un análisis y desarrollo de un simulador de banco, el cual permite realizar operaciones como depósito, retiro de saldo y la simulación de la atención en una cola de clientes. El análisis se basa en conceptos de programación estructurada, mediante un diagrama de programación modular, con el fin de garantizar claridad, eficiencia y simplicidad del código. Se evalúan aspectos clave como la estructura del pseudocódigo, la implementación del código fuente y el cumplimiento de principios de calidad que aseguren que el código sea entendible y eficiente. El sistema está implementado en Python y acompañado de un pseudocódigo en PSeInt que ayuda a visualizar la lógica.

Palabras clave: Banco, Simulador, sistema, programación, cola, Python. Programación

I. INTRODUCCIÓN

En este documento describe el desarrollo de un sistema básico de simulación de operaciones bancarias utilizando Python. El sistema permite gestionar una cuenta bancaria y simular la atención de varios clientes en cola. Se aplican las normas IEEE 830, 1016 y 829 para asegurar una buena especificación de requisitos, diseño modular y documentación de pruebas.

IEEE 830 se refiere a la especificación de requisitos, definiendo claramente lo que se espera que haga el software.

IEEE 1016 establece cómo debe ser documentado el diseño del software, asegurando que este sea comprensible y modular.

Documento entregado el 27 de mayo de 2025 Este trabajo fue apoyado por la clase de Programación 1, donde fueron adquiridos los conocimientos previos a la realización de este trabajo.

II. ANÁLISIS

A. Contexto

Este sistema simula un banco que atiende clientes en cola, permitiéndoles realizar operaciones básicas con una cuenta bancaria.

B. Población

Trabajadores del banco que puedan llevar un orden

C. Limitaciones y alcance

Las limitaciones y alcances que se pudieron identificar son las siguientes:

Limitaciones

- No maneja múltiples cuentas.
- Solo una cuenta bancaria común.
- No hay persistencia de datos.

Alcances

- Tiene una simulación básica de operaciones bancarias.
- Permite orden y supervisión de la atención al cliente en cola.
- Nos permite tener un control y validación del monto en la cuenta.

III. OBJETIVOS DEL ANÁLISIS

Los objetivos de este análisis son los siguientes:

A. Objetivo general

Desarrollar un sistema que simule operaciones bancarias básicas usando Python y PSeInt con atención a clientes en cola.

B. *Objetivos específicos*

- Diseñar y crear una clase para gestionar una cuenta bancaria.
- Simular la atención en clientes en una cola, para la organización y mejora en atención del banco.

- Validar los montos que sean depositados o retirados y realizar las respectivas operaciones bancarias conociendo un monto inicial..

IV. Desarrollo

1.1 Introducción:

Este documento especifica los requisitos para un sistema de simulación bancaria básico, con el objetivo de permitir a los usuarios realizar operaciones como depósito, retiro y consulta de saldo, además de simular la atención de clientes en una cola de servicio. El sistema busca ser una herramienta didáctica para comprender conceptos como gestión de cuentas, estructuras de datos (colas) y programación modular en Python y PSeInt.

1.2 Requisitos Funcionales:

- RF1: Permitir el ingreso de clientes a la cola de atención.
- RF2: Procesar operaciones bancarias básicas: depósito, retiro y consulta de saldo.
- RF3: Validar montos ingresados, evitando errores lógicos (como montos negativos o mayores al saldo).
- RF4: Mostrar mensajes de confirmación de operaciones exitosas o fallidas.
- RF5: Llevar control del saldo actualizado tras cada operación.
- RF6: Simular la atención secuencial de clientes (primero en entrar, primero en salir).
- RF7: Implementar la lógica mediante programación modular y estructurada.
- RF8: Generar un entorno de pruebas sencillo y comprensible para estudiantes principiantes en programación.

1.3 Requisitos No Funcionales:

- RNF1: El sistema debe ser confiable y sin errores lógicos.
- RNF2: Debe ser fácil de usar e interpretar por usuarios sin experiencia técnica..
- RNF3: El sistema debe permitir su ampliación futura para incluir múltiples cuentas u operaciones adicionales.
- RNF4: El sistema debe estar correctamente comentado para facilitar su mantenimiento.
- RNF5: El código debe cumplir principios básicos de claridad, eficiencia y buena estructuración.
- RNF6: El sistema debe ejecutarse con bajo uso de recursos, ideal para entornos educativos.
- RNF7: El sistema debe poder correr en múltiples plataformas (Windows, Linux, etc.) con Python instalado o en entornos de desarrollo como PSeInt.

2. IEEE 1016 (Descripción del Diseño del Software):

2.1 Arquitectura del Sistema:

El simulador está diseñado bajo una arquitectura monolítica y modular, con separación lógica entre las funciones que procesan operaciones bancarias y la simulación de atención de clientes:

- Capa de presentación: Interfaz basada en consola, tanto para PSeInt como Python.
- Capa de lógica de negocio: Encargada del manejo del saldo, operaciones bancarias y control de flujo.
- Capa de datos: Uso de variables locales para mantener el estado de la cuenta durante la ejecución. (Sin base de datos en esta versión.)

2.2 Diseño de Módulos

- Módulo de Cuenta Bancaria: Contiene la lógica para realizar depósitos, retiros y consultas de saldo. (Clase CuentaBancaria en Python).
- Módulo de Simulación de Cola: Gestiona la lista de clientes y ejecuta las operaciones según lo que el usuario elija.
- Módulo de Interfaz de Usuario: Solicita entrada de datos y muestra resultados en consola, de forma clara e intuitiva.
- (Opcional futuro) Módulo de Reporte: Para mostrar un resumen final de operaciones realizadas por cliente (no implementado aún).

V. Recomendaciones y Mejoras

- Modularización del código: Es recomendable separar la lógica de cuenta bancaria, la atención de clientes y la interfaz de usuario para mayor claridad y facilidad de prueba.
- Validaciones de entrada: Incluir validaciones más robustas para evitar entradas no numéricas o montos negativos en las operaciones.
- Extensibilidad: Se puede ampliar el sistema para soportar múltiples cuentas, transferencias entre cuentas y registro histórico de operaciones.
- Persistencia de datos: Implementar almacenamiento básico (por ejemplo, en archivos .txt o .csv) para mantener el historial de movimientos después de cerrar el programa.

- Interfaz gráfica: A futuro, podría implementarse una interfaz gráfica amigable usando Tkinter (en Python) o una interfaz web sencilla con Flask.VI.

VI. Diagrama de flujo

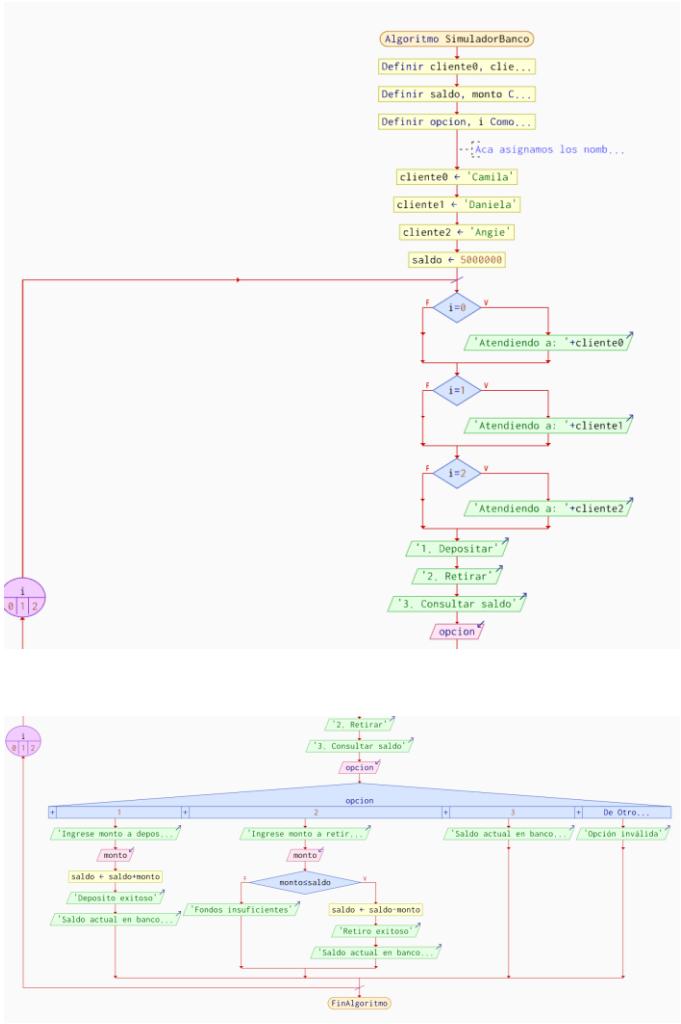

Fig. 1. Esta imagen es la representación gráfica del código, donde se tendrá una visualización del proceso que llevara a cabo; es el boceto a tener en cuenta en nuestro pseudocódigo realizado en pseint para un mejor desarrollo y entendimiento en el código principal presentado en Python

VII. Descripción del Código

1. Clase CuentaBancaria

Esta clase representa la cuenta principal del banco, que empieza con un saldo de $5.000.000.

__init__(self)

Constructor que inicializa el saldo de la cuenta bancaria con cinco millones de pesos.

python

CopiarEditar

self.saldo = 5000000

depositar(self, monto)

Método que permite aumentar el saldo si el monto ingresado es mayor que cero.

Si el monto es válido, se suma al saldo actual y se imprime un mensaje de éxito.

Si el monto es negativo o cero, se notifica al usuario que el valor no es válido.

retirar(self, monto)

Método que permite restar dinero del saldo si:

El monto es mayor a cero y

El saldo disponible es suficiente.

Si no se cumple alguna de estas condiciones, se muestran mensajes indicando el error (monto negativo o fondos insuficientes).

consultar_saldo(self)

Este método muestra por pantalla el saldo actual de la cuenta bancaria.

2. Función procesar_cola_clientes()

Esta función simula la atención en fila de varios clientes del banco.

Se crea una instancia de la clase CuentaBancaria.

Se define una lista con los nombres de tres clientes: "Camila", "Daniela" y "Angie".

Se usa un ciclo for para recorrer la cola de clientes.

A cada cliente se le muestra un menú con tres opciones:

- Depositar
- Retirar
- Consultar saldo

Dependiendo de la opción ingresada por el usuario, se llama al método correspondiente de la clase.

Al finalizar cada turno, se imprime un mensaje indicando el fin de la atención al cliente actual.

3. Programa principal

La función procesar_cola_clientes() se llama directamente para iniciar la ejecución del simulador.

VIII. PROPÓSITO:

Este análisis tiene como propósito desarrollar un simulador bancario básico que permita comprender el funcionamiento de una cuenta bancaria y la gestión de clientes en una cola de atención. El proyecto facilita el aprendizaje de estructuras como listas, condicionales, clases y funciones en Python, así como el uso de pseudocódigo y diagramas de flujo en PSeInt. Además, permite visualizar cómo se puede modelar un sistema real simplificado mediante programación estructurada y modular.

IX. CONCLUSIÓN

El desarrollo del simulador de banco demuestra que la metodología de resolución de problemas aplicada es efectiva para construir sistemas comprensibles y funcionales. Se logró implementar un algoritmo que permite simular operaciones bancarias básicas (depósito, retiro y consulta de saldo) y atención secuencial de clientes, fomentando el pensamiento lógico y la estructuración del código. La documentación siguiendo el formato IEEE permite organizar el proyecto de forma clara y profesional, facilitando su mantenimiento, prueba y mejora en el futuro.

X. REFERENCIAS

[1] 1. IEEE Standard for Software Test Documentation, IEEE 829-2008.
[2]
[3]
[4] 2. IEEE Recommended Practice for Software Requirements Specifications, IEEE 830-1998.
[5]
[6]
[7] 3. IEEE Standard for Information Technology—Systems Design, IEEE 1016-2009.