Universidad Mariano Galvez
Facultad de Ingeniería, Matemáticas y Ciencia Físicas
Ing. Michael Rodolfo Asturias L
Programación III

Proyecto # 1

Informe Tecnico

Integrantes:
0900-22-9686 | Guillermo Antonio Ortiz López
0900-22-149 | Amanda Lorena Garcia Castellanos

Guatemala, 4 de Mayo del 2024

Índice

Introducción

El presente informe técnico tiene como objetivo principal documentar el diseño e implementación de las estructuras de datos utilizadas en el sistema de gestión de tarjetas de crédito en este proyecto. Este documento proporcionará una visión detallada de cómo estas estructuras fueron concebidas y posteriormente integradas en el sistema, destacando su importancia y funcionalidad dentro del contexto del proyecto.

Diseño de las Estructuras de Datos

1) **Listas Enlazadas:** Se utilizaron listas enlazadas para implementar la funcionalidad de almacenamiento y gestión de datos de las tarjetas de crédito. Esta elección se basó en la flexibilidad y eficiencia de las listas enlazadas para la inserción y eliminación de elementos, lo que resulta útil en un contexto dinámico como el de las transacciones financieras.

2) **Pilas(Stacks):** Las pilas se emplearon para la implementación de la funcionalidad de gestión de solicitudes de aumento de límite de crédito. Optamos por utilizar pilas debido a su naturaleza de tipo LIFO (Last In, First Out), que se alinea con el proceso de procesamiento de solicitudes de manera secuencial y priorizada.

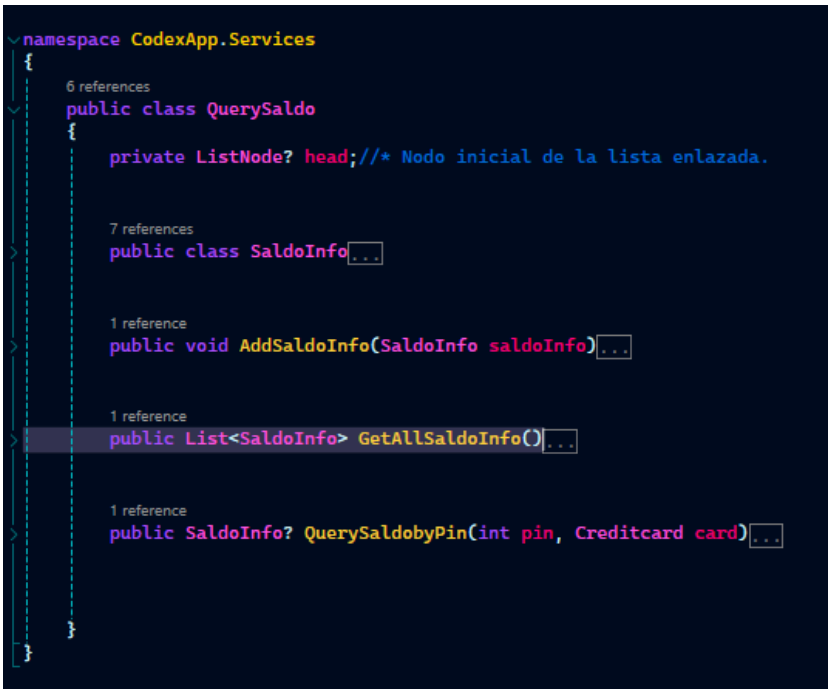
3) **Colas (Queues):** Las colas se utilizaron para gestionar y procesar los pagos realizados por los usuarios. La elección de utilizar colas se fundamentó en su comportamiento de tipo FIFO (First In, First Out), lo que garantiza un procesamiento justo y ordenado de los pagos entrantes.

Implementación

Clase “QuerySaldo”

La clase QuerySaldo se encarga de gestionar la información de saldo de las tarjetas de crédito mediante el uso de una lista enlazada.

1. **Atributos:** head: Representa el nodo inicial de la lista enlazada que almacena la información de saldo.
2. **Método AddSaldoInfo:** Este método permite agregar nueva información de saldo a la lista enlazada. Si la lista está vacía, se crea un nuevo nodo inicial; de lo contrario, se recorre la lista hasta el final y se agrega el nuevo nodo.
3. **Método GetAllSaldoInfo:** Este método devuelve una lista que contiene toda la información de saldo almacenada en la lista enlazada. Recorre la lista enlazada y agrega cada elemento a la lista de salida.
4. **Método QuerySaldobyPin:** Este método se utiliza para consultar la información de saldo de una tarjeta de crédito específica identificada por su número de PIN. Crea un nuevo objeto SaldoInfo con la información correspondiente y lo agrega a la lista enlazada mediante el método AddSaldoInfo.



Clase “PinUpdate”

La clase PinUpdate se encarga de gestionar las actualizaciones de PIN de las tarjetas de crédito mediante el uso al igual que la clase QuerySaldo, de una lista enlazada.

1. **Atributos:**
   - **notificationId:** Entero que representa el identificador de las notificaciones.
   - **head:** Representa el nodo inicial de la lista enlazada que almacena los cambios de PIN
2. **Método UpdatePins**: Este método se utiliza para actualizar el PIN de una tarjeta de crédito específica. Si la tarjeta coincide con el PIN proporcionado, se actualiza el PIN y se genera un evento PinUpdateEvent. Además, se crea un objeto PinChangeUpdate con los datos de la actualización y se agrega a la lista enlazada mediante el método AddPinChangeRequest.
3. **Método ProcessPinChanges**: Este método procesa los cambios de PIN almacenados en la lista enlazada. Recorre la lista y genera una notificación para cada cambio de PIN encontrado.
4. **Método AddPinChangeRequest**: Este método se utiliza para agregar un cambio de PIN a la lista enlazada. Si la lista está vacía, se crea un nuevo nodo inicial; de lo contrario, se recorre la lista hasta el final y se agrega el nuevo nodo.
5. **Método GetAllPinChangeRequests:** Este método devuelve una colección de todos los cambios de PIN almacenados en la lista enlazada.

```
namespace CodexApp.Services
{
    6 references
    public class PinUpdate
    {
        private int notificationId = 0;

        public event EventHandler<PinUpdateEvent>? PinUpdateEv;

        private PinNode? head;

        0 references
        public PinUpdate()...

        1 reference
        public void UpdatePins(int pin, int updatedPin, Creditcard card)...


        1 reference
        public void ProcessPinChanges()...


        1 reference
        protected virtual void OnPinUpdate(int pin, int updatedPin)...

        1 reference
        public void AddPinChangeRequest(PinChangeUpdate pinChange)...


        1 reference
        public IEnumerable<PinChangeUpdate> GetAllPinChangeRequests()...
    }
}
```

Clase “PagoQueue”

La clase “PagoQueue” se encarga de administrar los pagos de las tarjetas de crédito mediante el uso de una cola.

Atributos:

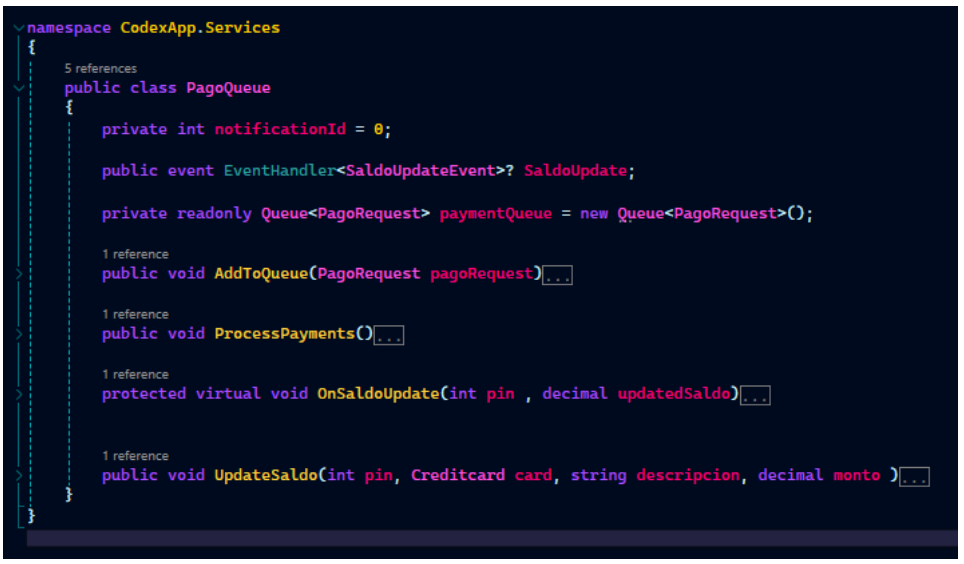
- **NotificationId:** Entero que representa el identificador de las notificaciones
- **PaymentQueue**: Cola que almacena los objetos PagoRequest, que contienen la información de los pagos pendientes.

Método AddToQueue: Este método se utiliza para agregar un objeto PagoRequest a la cola de pagos pendientes.

Método ProcessPayments: Este método procesa los pagos pendientes en la cola. Itera sobre la cola y, para cada objeto PagoRequest, genera una notificación con la descripción y el monto del pago realizado.

Método UpdateSaldo: Este método se utiliza para actualizar el saldo de una tarjeta de crédito después de realizar un pago. Si la tarjeta coincide con el PIN proporcionado, se suma el monto del pago al saldo de la tarjeta. Luego, se genera un evento SaldoUpdate con el nuevo saldo y se agrega un nuevo objeto PagoRequest a la cola de pagos pendientes.

Método OnSaldoUpdate: Este método se utiliza para invocar el evento SaldoUpdate con los datos actualizados del saldo de la tarjeta de crédito.

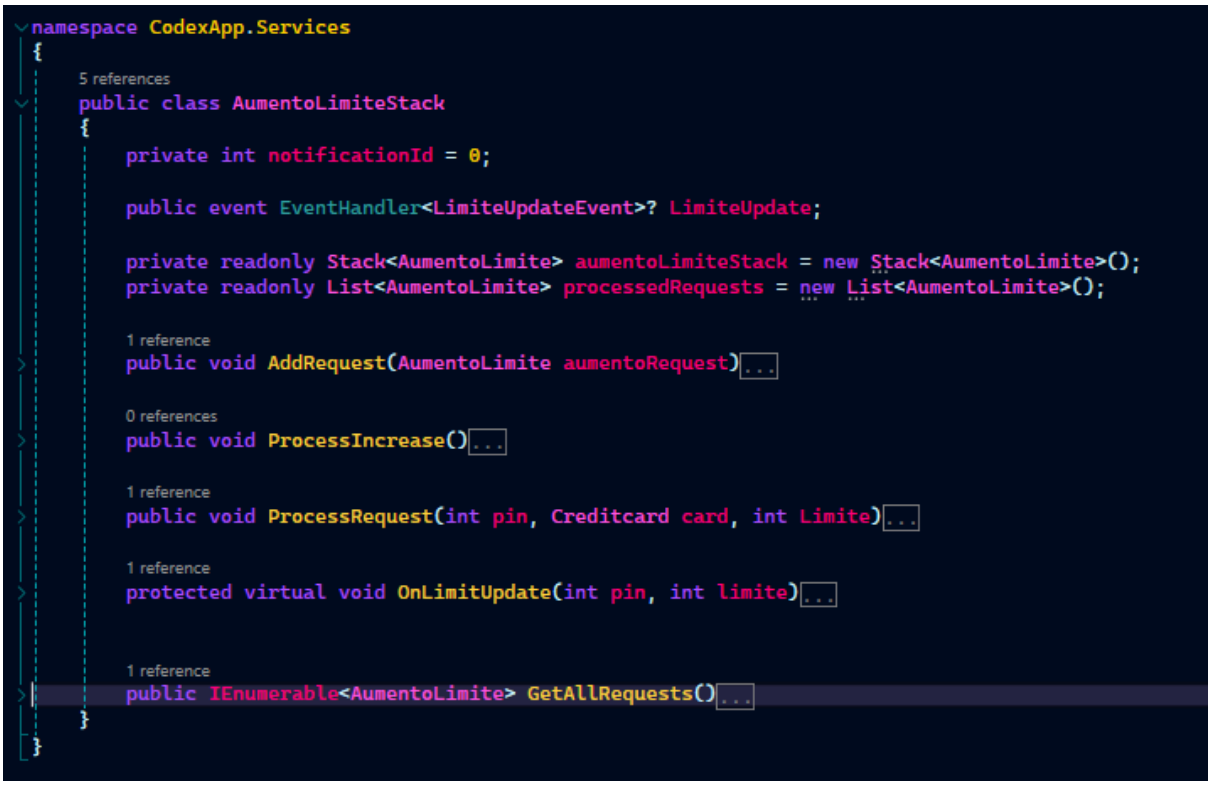


Clase “AumentoLimiteStack”

La clase "AumentoLimiteStack” se encarga de gestionar las solicitudes de aumento de límite de las tarjetas de crédito mediante el uso de una pila.

1. **Atributos:**
   - **NotificationId:** Entero que representa el identificador de las notificaciones.
   - **AumentoLimiteStack:** Pila que almacena las solicitudes de aumento de límite pendientes.
   - **ProcessedRequests**: Lista que almacena las solicitudes de aumento de límite procesadas.
2. **Método AddRequest**: Este método se utiliza para agregar una solicitud de aumento de límite a la pila de solicitudes pendientes.
3. **Método ProcessIncrease:** Este método procesa las solicitudes de aumento de límite pendientes en la pila. Itera sobre la pila y, para cada solicitud, genera una notificación con el nuevo límite establecido. Luego, agrega la solicitud a la lista de solicitudes procesadas y vacía la pila.
4. **Método ProcessRequest:** Este método se utiliza para procesar una nueva solicitud de aumento de límite. Si la tarjeta coincide con el PIN proporcionado, se actualiza el límite de la tarjeta con el nuevo valor especificado. Además, se genera un evento LimiteUpdate con el nuevo límite y se agrega la solicitud a la pila de solicitudes pendientes.
5. **Método OnLimitUpdate:** Este método se utiliza para invocar el evento LimiteUpdate con los datos actualizados del límite de la tarjeta de crédito.

6. **Método GetAllRequests:** Este método devuelve todas las solicitudes de aumento de límite, tanto pendientes como procesadas, en forma de una lista combinada.



```
namespace CodexApp.Services
{
    5 references
    public class AumentoLimiteStack
    {
        private int notificationId = 0;

        public event EventHandler<LimiteUpdateEvent>? LimiteUpdate;

        private readonly Stack<AumentoLimite> aumentoLimiteStack = new Stack<AumentoLimite>();
        private readonly List<AumentoLimite> processedRequests = new List<AumentoLimite>();

        1 reference
        public void AddRequest(AumentoLimite aumentoRequest)...

        0 references
        public void ProcessIncrease()...

        1 reference
        public void ProcessRequest(int pin, Creditcard card, int Limite)...

        1 reference
        protected virtual void OnLimitUpdate(int pin, int limite)...


        1 reference
        public IEnumerable<AumentoLimite> GetAllRequests()...
    }
}
```

Conclusión

En conclusión, se ha demostrado el uso fundamental de las estructuras de datos en este proyecto, destacando su importancia en la gestión eficiente de la información relacionada con las tarjetas de crédito y las operaciones financieras. La implementación de listas enlazadas, pilas y colas ha permitido un manejo ágil y organizado de los datos, asegurando un funcionamiento eficiente y fiable del sistema en su conjunto. Estas estructuras han sido clave para optimizar la gestión de saldo, procesar solicitudes de aumento de límite y gestionar los pagos de manera secuencial, demostrando su relevancia en el desarrollo de aplicaciones robustas y escalables.