2020

TAREA INTEGRADORA

METODO DE LA INGENIERIA Y REQUERIMIENTOS

SANTIAGO HURTADO SOLIS A003625, CRISTIAN MORALES A00328064, JUAN SEBASTIAN MORALES A00365920

Enunciado

Un gran banco desea desarrollar un software que modele el funcionamiento de una de sus sedes con mayor flujo de personas. Para ello, lo ha contratado a usted y a su equipo de trabajo con el objetivo de construir un programa capaz de solventar todas las necesidades del cliente, entre las cuales se encuentran el proceso de turnos al momento del ingreso (fila para clientes y fila para personas con diferentes prioridades), manejo de tablas de datos, entre otros.

Con el fin de llevar a cabo procesos estadísticos y de agilizar el servicio de atención, esta sede bancaria registra el nombre y la cédula de todos los usuarios que ingresan al establecimiento a la hora de obtener su turno. Con ello no sólo se busca en cuál de las tres filas ubicar a la persona, sino que permite que la persona encargada de la atención de dicho usuario pueda buscarlo de manera eficiente en la base de datos y obtener toda su información antes de que este llegue a su despacho. Los datos que encontrará el encargado serán: nombre, cédula, cuenta bancaria, tarjetas de débito/crédito, fecha de pago de la tarjeta de crédito y fecha en que se incorporó al banco. Dicha información deberá mostrarse en pantalla.

Una vez sea el turno de atender al usuario, éste podrá realizar una o más de las siguientes operaciones:

a) **Retiro/consignación:** el usuario podrá modificar el monto de su cuenta de ahorros al solicitar un retiro o consignación.

b) **Cancelación de cuenta:** borra sus datos de la base de datos de clientes y los incorpora a una exclusiva para aquellos que desertan de dicho banco. Asimismo, se guardará tanto la fecha como el motivo de cancelación.

c) **Pago de tarjeta:** el usuario podrá pagar el monto utilizado con la tarjeta de crédito hasta el momento. Puede realizar el pago en efectivo o a través de su cuenta de ahorros.

La información de todos los usuarios presentes en la sede bancaria deberá visualizarse en una tabla tipo hoja de cálculo y podrá ser organizada respecto a cuatro (4) parámetros de su escogencia. Con el objetivo de encontrar el algoritmo óptimo para el problema, el gerente le ha solicitado implementar un método de ordenamiento de su parecer por cada uno de los parámetros escogidos, con la restricción de que solamente uno (1) de ellos puede tener complejidad temporal promedio de (n^2).

Teniendo en cuenta los errores humanos que se introducen por parte de los cajeros ya sea por equivocaciones propias al digitar la solicitud del cliente, se le solicita al equipo de ingenieros agregar una funcionalidad de *undo* para poder deshacer las equivocaciones, incluso después de haberlas guardado.

El programa esperado por el banco debe implementar:

1. Una interfaz gráfica de usuario donde se pueda visualizar el estado actual de las 2 filas de espera.
2. Una interfaz gráfica de usuario que le permita visualizar y ordenar una tabla tipo hoja de cálculo por los siguientes criterios:
    a. Nombre del cliente
    b. Cédula del cliente
    c. Tiempo de vinculación del cliente
    d. Monto

    De los 4 algoritmos de ordenamiento solo uno puede ser de complejidad temporal promedio de O(n^2), los 3 restantes deben ser de mejor complejidad.

3. Una interfaz gráfica de usuario que le permita buscar la información bancaria de un cliente a partir de su cédula y realizar las siguientes acciones:
    a. Retiro: Aumentar el monto de la cuenta de ahorros
    b. Consignación
    c. Cancelación de cuenta
    d. Pago de la tarjeta
    e. *Undo* de una de las anteriores acciones

Usted debe utilizar el método de la ingeniería para resolver este problema y dejar evidencia en su informe de los resultados de cada fase. Por ejemplo, en la fase 1 deben identificar claramente el problema, justificarlo y especificar los requerimientos funcionales. Recuerde revisar el Resumen del Método de la Ingeniería y el ejemplo del Método de la Ingeniería aplicado a un problema.

Informe metodo de la ingeniería.

Entregables.

1. Informe PSP0. Cada estudiante debe entregar el informe de su desarrollo.
2. Entrega de informe del método de la ingeniería.
3. Análisis de complejidad temporal de cada uno de sus algoritmos
4. Análisis de complejidad espacial de cada uno de sus algoritmos
5. Especificación de Requerimientos y Diseño.
6. Diseño del TAD para cada estructura de datos requerida.
7. Diseño del diagrama de clases desacoplado y utilizando generics.

8. Diseño de los casos de prueba. Adicionalmente debe explicar cómo se resuelven dos casos, paso a paso (con dibujos, si es necesario), por cada estructura de datos diseñada e implementada.

9. Diseño del diagrama de clases de pruebas unitarias automáticas.

10. Todos los archivos deben estar almacenados en GitHub y debe evidenciarse su uso desde el inicio del proyecto.

11. Implementación de las pruebas unitarias automáticas.

Requerimientos funcionales:

El sistema debe estar en la capacidad de:

Asignar un turno a un usuario, con su nombre ,cédula y tipo de cliente si este es prioritario o normal al entrar al banco.

Registrar el nombre ,cedula , cuenta bancaria, tarjetas de débito/crédito, fecha de pago de la tarjeta de crédito y fecha en que se incorporó al banco , de los usuarios que ingresan al banco.

Mostrar la información del usuario del turno actual en la pantalla.

Mostrar el estado actual de las filas tanto de la normal como de la prioritaria, con la posibilidad que me permita avanzar de turno cuando termine el servicio.

Modificar el monto de la cuenta al solicitar consignación o retiro.

Realizar un retiro (sacar dinero del banco) teniendo en cuenta que el dinero máximo a sacar debe estar en un tope, además en caso de que no tenga suficientes fondos en la cuenta debe .

Realizar una consignación (enviar dinero a otra cuenta o a la misma cuenta), la cuenta o tarjeta debe tener fondos disponibles para poder realizar dicha función.

Cancelar una cuenta, borrando los datos de la base de datos y anexarlos a otra base de datos de clientes que dejan el banco. guardando así el motivo de cancelación

Pagar el monto de dinero utilizado por la tarjeta de crédito de una usuario, este proceso se puede hacer en efectivo, o con la cuenta de ahorros del usuario.

Buscar la información de los usuarios de la sede bancaria con su nombre, cédula,cuenta bancaria y tarjeta débito o crédito, en una hoja de cálculo, la cual se deberá visualizar en pantalla. Estos datos se podrán ordenar usando un método de ordenamiento diferente para cada uno.

Deshacer las últimas modificaciones realizadas por el usuario, incluso si estas ya han sido guardadas.

Requerimientos No funcionales:

Permitir ordenar una hoja de cálculo con respecto a 4 parámetros, cada uno de estos parámetros debe tener su propio algoritmo de ordenamiento, pero solo uno de ellos puede tener una complejidad de O(n^2) todos los demás deben tener una complejidad menor.

Desarrollar interfaces atractivas e intuitivas para el usuario, las cuales permitan mostrar las acciones y operaciones a llevar a cabo.

Realizar una búsqueda eficiente a la hora de buscar un usuario por cédula, mostrando la información correcta de este.

Método de la ingeniería

Paso 1:

Identificación de necesidades:

-Software para solventar las necesidades del cliente

-LLevar registro de los procesos estadísticos

-Agilizar el servicio de atención

-Solución eficiente en cuanto al manejo de información de los clientes

-El sistema le debe permitir al usuario realizar funciones como retiro,consignación,cancelación de cuenta.

-Encontrar algoritmos óptimos de búsqueda

-Mostrar el estado de filas

-Mostrar la información del cliente (nombre, cuenta, tarjeta, id, etc)

-Al eliminar un cliente guardarlo en otra base de datos.

Definición del problema

Nuestro cliente necesita modelar el funcionamiento de una de sus sedes, en las funcionalidades de la seda se encuentra que tiene un sistema de turnos el cual tiene dos turnos por separado uno para clientes, y el otro para personas con prioridades, en estos turnos se desplegará cierta información necesaria para que el proceso de atención sea más ágil y se pueda tener un proceso estadístico. Una vez realizado el proceso de turno el usuario del turno podrá: realizar un retiro o una consignación, la cual modificará el monto de su cuenta; tambien podra cancelar su cuenta, lo cual borra su cuenta pero se guardará datos como la fecha y el motivo de la cancelación; y por último podrá pagar el monto de su tarjeta de crédito hasta ese momento. También el cliente necesita que la información de los usuarios del sistema se pueda organizar en una especie de hoja de cálculo, la cual se puede ordenar de acuerdo a 4 parámetros distinto, pero estos cuatro parámetros deben estar ordenados por 4 algoritmos diferentes, entre los cuales uno y solamente uno puede tener una complejidad de O(n^2) los demás deben ser menores a este, y por último el cliente necesita que el programa incluye una implementación de la opción *undo* es decir que pueda volver a la última modificación realizada.

Paso 2:

Recopilacion de informacion

Para realizar este trabajo, además de nuestros conocimientos previos necesitamos investigar sobre qué algoritmos de ordenamientos usar, los cuales nos sean útiles a la hora de implementarlos, en nuestra investigación encontramos los siguientes 7 algoritmos con su respectiva complejidad:

- **Burbuja (Bubble Sort):** Complejidad O(n^2)
- **Conteo (Counting Sort):** Complejidad O(n+k)
- **Montones (Heapsort):** Complejidad O(n log n)
- **Inserción (Insertion Sort):** Complejidad O(n^2)
- **Mezclas (Merge Sort):** Complejidad O(n log n)
- **Rápido (Quicksort):** Complejidad O(n log n)
- **Selección (Selection Sort):** Complejidad O(n^2)

El funcionamiento de los algoritmos a utilizar es el siguiente:

Selection:

Descripción del algoritmo

El ordenamiento por selección (Selection Sort en inglés) es un algoritmo de complejidad O(n^2) Este algoritmo de clasificación es un algoritmo basado en comparación, en el que la lista se divide en dos partes, la parte ordenada en el extremo izquierdo y la parte no ordenada en el extremo derecho. Inicialmente, la parte ordenada está vacía y la parte no ordenada es la lista completa.

Su funcionamiento es el siguiente:

- Buscar el mínimo elemento de la lista.
- Intercambiarlo con el primero.
- Buscar el siguiente mínimo en el resto de la lista.
- Intercambiarlo con el segundo.

Heapsort:

Este algoritmo consiste en almacenar todos los elementos en un montículo y luego extraer el nodo que queda como raíz en iteraciones sucesivas obteniendo el conjunto ordenado. Para esto el método realiza los siguientes pasos:

- Se construye el Heap/montículo a partir del arreglo original.
- La raíz se coloca en el arreglo.
- El último elemento del montículo se vuelve la raíz.
- La nueva raíz se intercambia con el elemento de mayor valor de cada nivel.
- Tras el paso anterior la raíz vuelve a ser el mayor del montículo.
- Se repite el paso 2 hasta que quede el arreglo ordenado.

Mergesort

El algoritmo de ordenamiento por mezcla (*merge sort* en inglés) es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Es de complejidad O(n log n).

Conceptualmente, el ordenamiento por mezcla funciona de la siguiente manera:

1. Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso:
2. Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
3. Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.
4. Mezclar las dos sublistas en una sola lista ordenada.

El ordenamiento por mezcla incorpora dos ideas principales para mejorar su tiempo de ejecución:

1. Una lista pequeña necesitará menos pasos para ordenarse que una lista grande.
2. Se necesitan menos pasos para construir una lista ordenada a partir de dos listas también ordenadas, que a partir de dos listas desordenadas. Por ejemplo, sólo será necesario entrelazar cada lista una vez que están ordenadas.

QuickSort

Es un método de ordenamiento que se basa en la técnica de divide y vencerás, resulta más fácil ordenar listas pequeñas que una grande , con lo cual irá descomponiendo la lista en dos partes y ordenando esas partes

- Dada una lista, elegir uno de sus elementos, que llamamos **pivot**
- Dividir la lista en dos sublistas:
  - una con los elementos "menores"
  - otra con los elementos "mayores"
- Ordenar recursivamente ambas sublistas
- Armar la lista resultado como: menores Ordenados + pivot + mayores Ordenados

QUEUE Y PRIORITY QUEUE

Queue es una interfaz que extiende a Collection y proporciona operaciones para trabajar con una cola. PriorityQueue es una de las clases que implementa esta interfaz y ordena los elementos en base a su orden natural, según lo especificado por el método comparó los elementos Comparable, o mediante un objeto Comparator que se suministra a través del constructor. Esta clase proporciona una funcionalidad que permite inserciones en orden y eliminaciones de la parte frontal (desencolar normalmente). Al insertar se hace según una prioridad, de tal forma que el elemento de mayor prioridad se sitúa el primero. Las operaciones más habituales son offer para insertar en la posición adecuada según la prioridad, poll por desencolar, peek para obtener el primero de la cola, clear para eliminar todos los elementos de la cola y size para obtener el número de elementos de la cola .

STACK

La clase Stack extiende a la clase Vector para implementar una estructura de datos de tipo pila. Funciona como las pilas que conocéis y básicamente tiene los métodos push (apilar), pop (desapilar), peek (cima) y empty (está vacía).

La Stack representa una pila de objetos de último en entrar, primero en salir (LIFO). Extiende la clase Vector con cinco operaciones que permiten tratar un vector como una pila.

Cuando se crea una pila por primera vez, no contiene elementos.

Hash table

La tabla hash es una estructura de datos que representa datos en forma de pares clave-valor. Cada clave se asigna a un valor en la tabla hash. Las claves se utilizan para indexar los valores/datos. Un enfoque similar es aplicado por un array asociativo.

Los datos se representan en un par de valores clave con la ayuda de claves como se muestra en la siguiente figura. Cada dato está asociado con una clave. La clave es un entero que apunta a los datos.

Aplicaciones de tabla de hash

Las tablas de hash se implementan donde

- se requiere búsqueda e inserción de tiempo constante
- aplicaciones criptográficas
- se requieren datos de indexación.

Montículos.

Heap es un caso especial de estructura de datos de árbol binario balanceada donde la clave del nodo raíz se compara con sus hijos y se organiza en consecuencia. Si α tiene un nodo hijo β entonces -

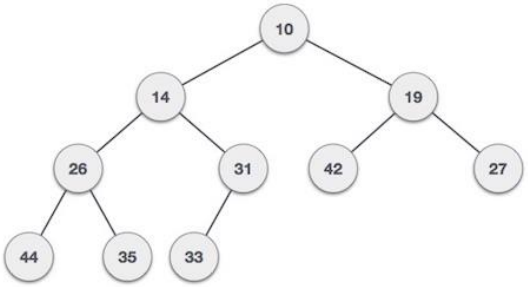
tecla (α) ≥ tecla (β)

Como el valor de parent es mayor que el de child, esta propiedad genera Max Heap . Según este criterio, un montón puede ser de dos tipos:

Entrada (A)→ 35 33 42 10 14 19 27 44 26 31

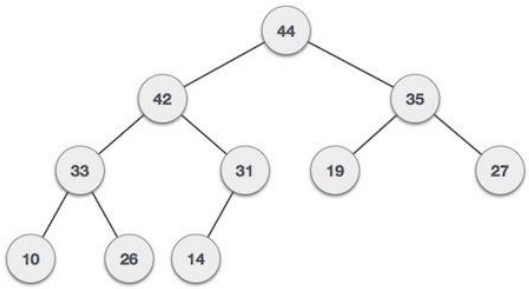
Min-Heap : donde el valor del nodo raíz es menor o igual que cualquiera de sus hijos.

10,14,19,26,27,31,33,35,42,44



Max-Heap : donde el valor del nodo raíz es mayor o igual que cualquiera de sus hijos.

44,42,35,33,31,27,26,19,14,10



Ambos árboles se construyen utilizando la misma entrada y orden de llegada.

Consignación: la acción de consignar, destinar o designar una cantidad de dinero para efectuar el pago gastos o deudas.

Retiro: hace referencia a la acción de extraer dinero en efectivo de un banco. Para que este proceso sea posible, la persona debe contar con una cuenta en la entidad bancaria en cuestión y, a la vez, tener fondos disponibles en la misma.

Complejidad temporal: En informática, la complejidad temporal es la complejidad computacional que describe la cantidad de tiempo que lleva ejecutar un algoritmo. La complejidad temporal se estima comúnmente contando el número de operaciones elementales realizadas por el algoritmo, suponiendo que cada operación elemental requiere una cantidad fija de tiempo. Por lo tanto, la cantidad de tiempo necesario y el número de operaciones elementales realizadas por el algoritmo difieren en un factor constante como máximo.

Complejidad espacial

Esta sección se enfoca en el uso de memoria (usualmente RAM) por los algoritmos mientras son ejecutados. Así como la complejidad temporal, explicada anteriormente, parte del análisis de un algoritmo se hace vía la complejidad espacial para obtener un estimado del uso de memoria principal expresado mediante una función según el tamaño de la entrada. El resultado es expresado usualmente en notación O grande.

Existen 4 aspectos relevantes a considerar:

- La cantidad de memoria requerida por el código del algoritmo.
- La cantidad de memoria requerida para almacenar los datos de entrada.
- La cantidad de memoria requerida para los datos de salida (algoritmos como los de ordenación suelen reorganizar los datos de entrada y por ello no necesitan memoria extra para la salida).
- La cantidad de memoria requerida en cuanto a espacio de trabajo del algoritmo para realizar los cálculos y asignaciones (tanto para variables como cualquier espacio necesario en la pila para almacenar llamadas a subrutinas, este espacio es particularmente significativo para algoritmos que utilizan técnicas recursivas).

Paso 3 Búsqueda de soluciones creativas

Se plantean las siguientes soluciones mediante una lluvia de ideas para realizar el software, enfocándonos en el análisis, diseño herramientas a utilizar en la implementación de este. Las soluciones aquí se limitan a ser meramente tecnológicas, puesto que decidimos plantear una solución respectiva de software, una de ellas es mediante recursos a utilizar como los ordenamientos, pilas, colas, y demás, además de una interfaz la cual posibilite observar todos estos cambios y funciones del programa.

1. Una solución que pensamos a la hora de diseñar nuestra tarea integradora fue modelar el problema en clases, utilizando herencia, y demás, una a la que llegamos fue banco como nuestra controladora, y otras que identificamos con sus atributos y respectivos métodos, fue la clase tarjeta, cliente y turno, utilizando la librería calendario de java para el manejo de fechas que nos piden en los requerimientos, por otro lado en esta solución pensamos en implementar las estructuras de datos genéricas desde 0, entre ellas están las pilas, colas, colas de prioridad, hashtable, diccionarios, las cuales vamos a utilizar en el desarrollo de las funciones que nos pide nuestra aplicación, tales como la gestión y el manejo de usuarios con los turnos, donde una opción eficiente es usar una tabla hash, las filas del banco mediante la queue, y la priorityqueue, la función de undo que podemos desarrollar con una stack, guardando las funciones a realizar, y en caso de cambios, se saca esta de la pila. otro punto en esta solución es los algoritmos de ordenamiento que vamos a utilizar para ordenar la información por criterios de los clientes los cuales tienen que ser de complejidad mayor a O (n^2), solo uno puede ser así, entonces planteamos el quick sort, merge sort, heap sort y el sélection siendo este de complejidad temporal mencionada anteriormente.
aquí también planteamos el uso para la interfaz gráfica de la librería de javafx, donde podemos desarrollar las interfaces mediante scene builder, y personalizarlas mediante un archivo en formato css.
Acompañando estas interfaces,su diseño y uso sería todo en diferentes ventanas las cuales muestre la selección de turnos, buscar una cédula y mostrar información de los usuarios, y los usuarios se vean en la tabla, para organizarlos.

2.En esta solución, se modela de manera más completa el diagrama de clases, banco como la clase controladora, cuenta, tarjeta de crédito, tarjeta de ahorros, cliente normal, cliente prioritario y turno, utilizando también la librería calendar para el manejo de fechas, en esta solución las estructuras de datos a utilizar se utilizan mediante las distintas librerías de java, como stack, priority queue, queue, y demas. con la diferencia de manejar un arraylist de usuarios, y las librerías de java. añadiendo esta solución mostrará un reporte con los usuarios eliminados.
los algoritmos de ordenamiento a utilizar en esta solución, planteamos el counting sort , selection , quicksort, mergesort. Por otro lado, la ejecución del código se mostrará en consola.

3. Esta solución la planteamos modelando un diagrama de clases, donde la controladora será sede Banco, sus otras clases seria, cuenta, cliente, turno, e implementando una clase para las fechas a utilizar, la cual tendría, dia,mes y año.
Aquí se implementan las Queue y priority queue desde 0, y utilizando la hashtable, stack, del api de java, por otro lado esta solución nos daría reportes de los usuarios agregados junto a estos las cuentas, además de utilizar listas enlazadas para los turnos que van asignados a los usuarios.
otro punto clave aquí, es que se plantean unas interfaces diseñadas en scene builder, pero sencillas sin estilos css, ni nada por el estilo, los ordenamientos a utilizar en esta solución son el counting sort, merge sort, quicksort y heapsort.

4.Otra Solución planteada fue realizar el diagrama de clases, modelando el banco, como la controladora, usuarios, fila de prioridad, fila normal, tarjeta de crédito, tarjeta de ahorros, turnos, siendo este un modelado muy particionado, por otro lado se usaron algoritmos de ordenamiento los cuales conocemos como merge sort, quicksort, heapsort, y el bubblesort,cumpliendo así con los requerimientos, también se piensa en el diseño , desarrollo e implementación de la interfaz mediante java swing, una librería gráfica la cual permite el manejo de muchos componentes gráficos, por otro lado, las estructuras a usar, serian del api de java, para el uso de sus funciones a implementar, en cada uno de los requerimientos, por otro lado en esta solución se planea, conectar las bases de datos mediante MYSQL, para la optimización, guardado y búsqueda de las tarjetas, y usuarios.

5. Esta solución es un tanto parecida a la anterior, pero, cuenta con un modelado, el cual tenemos como controladora, banco, usuario, cuenta, tarjeta, y turno, contando con estructuras propias del api de java, y algunas implementadas por el equipo de desarrollo, por otro lado, los ordenamientos se piensan realizar mediante la tabla de java fx, el cual les aplica un ordenamiento a los datos. Por otro lado, el desarrollo de la interfaz gráfica sería mediante javafx, con una opción en especial de permitir al usuario sacar un reporte de su cuenta.

Paso 4. Transición de las Ideas a los Diseños Preliminares

1. Es una solución completa, y atractiva para el usuario, además de cumplir con todos los requerimientos, implementación propia de estructuras de datos, complejidad adecuada de los algoritmos de ordenamiento, presenta fallas en su modelamiento, pero utiliza un modelo el cual da una óptima y eficiente solución al problema, destacando un uso de interfaz intuitiva para el usuario, además de que le podemos añadir mejoras a la hora de su implementación, tales en el diagrama, añadirle otras clases manejando herencia, realizar las pruebas automáticas a cada estructura implementada, por otro lado, se manejan excepciones personalizadas para ciertos casos en el programa.

2. Esta solución presenta rasgos buenos y distintivos, además del uso de reportes y demás, pero la implementación se queda corta, por ser más factible, ya que usa herramientas del api de java, los ordenamientos cumplen con la con el requerimiento dado, pero su ejecución en consola disminuye las posibilidades a la hora de la usabilidad del usuario, y no mostraria las tablas que me piden en los requerimientos, además su modelamiento es complejo, donde se pueden utilizar ciertos elementos como tipos de clientes , sin necesidad de crear un cliente de cada tipo, lo mismo pasa con las tarjetas.

3. Esta solución tiene un modelamiento el cual se puede plantear mejor usando librerías para ciertas funciones, su implementación es mixta tanto propias y del api de java, siendo esto algo eficiente para algunas funciones, pero no cumple con los objetivos planteados del proyecto, además de que sus algoritmos de ordenamiento tienen una complejidad mayor a las demás soluciones. Por otro lado el diseño y uso de interfaces puede ser interesante, pero sencillo, lo cual puede ocasionar confusión
   a la hora de que el usuario lo utilice.

4. Solución práctica y rápida a la hora de su implementación,presenta fallos en su modelamiento, por tener tantas clases las cuales se pueden dividir, sumado a eso la desventaja de que la librería java swing, está quedando obsoleta a pesar de su gran capacidad de funciones, posee una complejidad mayor a la hora de implementar una base de datos, lo cual aun el equipo de desarrollo no conoce, siendo esto una muy buena funcionalidad para el registro ordenado de datos, pero la desventaja de no conocer su implementación, además de que tampoco cumple con la implementación propia de estructuras de datos.

A continuación se muestran las soluciones organizadas por los criterios que se plantearon a la hora de pensar el problema, como se modela, implementación de estructuras de datos, diseño y uso de interfaces, además de los algoritmos de ordenamiento a utilizar.

Posteriormente se descartan, y moldeamos las ideas para obtener una solución conforme a los requerimientos.
Escogimos 3 opciones las cuales fuimos desglosando en los criterios mencionados anteriormente

Diseño de diagrama de clases

Aquí se plantea cómo podemos modelar el problema mediante clases, las cuales podemos relacionar con el mundo real.

Diseño de las diferentes soluciones al realizarlos en un diagrama de clases.

1. Clases:banco, tarjeta,cliente,turno, estructuras de datos.
2. clases: banco, cuenta, tarjeta de crédito, tarjeta de ahorros, cliente normal, cliente prioritario, turno.
3. clases: Banco, cuenta, cliente, turno, día, mes, año.

Interfaces:

Aquí planteamos soluciones de cómo desarrollar las interfaces de nuestra tarea.

1. Uso de la librería de java fx, en donde podemos diseñar las interfaces como un archivo fxml, en scene builder, y aplicarle estilos de personalización en formato css.

2. El uso de la librería gráfica java swing, es una opción ya que nos permite trabajar con una librería la cual tiene tiene acceso a diversas funciones, y su uso es muy intuitivo.

3. diseñar las interfaces sencillas y funcionales mediante scene builder en formato Fxml.

Estructuras de datos a usar:

En el siguiente criterio, se plantean soluciones para el uso de diferentes estructuras de datos, todo esto para la implementación de funciones en la aplicación.

1. Estructuras a implementar propias.
   Manejo de cola de prioridad para el uso de fila de prioridad de los usuarios.
   Manejo de la pila para guardar las opciones, y ejecutar el botón undo.
   Manejo de la cola para guardar los usuarios por atender en la fila normal.
   Manejo de hashtable para guardar la información de usuarios y agregarlos en esta estructura, además de los turnos.

2. Uso del api de java para implementarlas.
   Manejo de cola de prioridad para las filas a atender.
   Manejo de la pila para guardar las opciones, y ejecutar el botón undo.
   Manejo de BST para agregar los usuarios

Manejo de Arraylist para los turnos.
Manejo de linkedlist para las tarjetas.

3. Uso del api de java y estructuras a implementar propias.
   Manejo de la fila de prioridad mediante una lista
   Manejo de una cola para guardar las opciones de undo
   Manejo de un arreglo para el registro de
   usuarios
   Manejo de un arraylist para los turnos
   Manejo de un arreglo para las cuentas a registrar
   Manejo de un Abb para la tabla de datos del cliente.

Algoritmos de ordenamiento

Como en el problema nos plantean es necesario escoger 4 algoritmos de ordenamiento para organizar por criterios la información de los usuarios, al lado su complejidad temporal.

1.
Quicksort O(n log n)
selection O(n^2)
mergesort O(n log n)
Heapsort O(n log n)

2.
 counting sort O(n+k)
selection O(n^2)
quicksort O(n log n)
mergesort O(n log n)

3.
mergesort O(n log n)
búsqueda lineal O(n)
quicksort O(n log n)
burbuja O(n^2)

Forma de visualización de app

1. Mostrar la interfaz de usuario en diferentes ventanas tanto así que vea el cliente y el trabajador del banco.
2. Poder mostrar la interfaz de usuario cargada toda en una sola pantalla.
3. mostrar únicamente las 3 interfaces que me piden.

Paso 5. Evaluación y Selección de la Mejor Solución

Escogimos 3 opciones las cuales nos parecieron viables, e integrales.

Factible: la solución es fácil de implementar, diseñar y analizar a la hora de realizarla.
ESCALA
1 muy fácil
2 fácil
3 medio
4 difícil
5 muy difícil

Eficiente: La solución cumple con la realización de todos los requerimientos.
ESCALA
1 ninguno
2 algunos
3 medio
4 casi todos
5 todos

Complejidad: La solución, conjunto , a su diseño y la implementación que grado de dificultad presenta.
ESCALA
1 muy facil
2 facil
3 medio
4 difícil
5 muy difícil

Integral: La solución abarca todos los requerimientos a realizar.
ESCALA
1 ninguno
2 algunos
3 medio
4 casi todos
5 todos

Usabilidad y manejo: La solución a la hora de probarlo el cliente es fácil de usar.
ESCALA

1. confuso
2. medio

3. ordenado

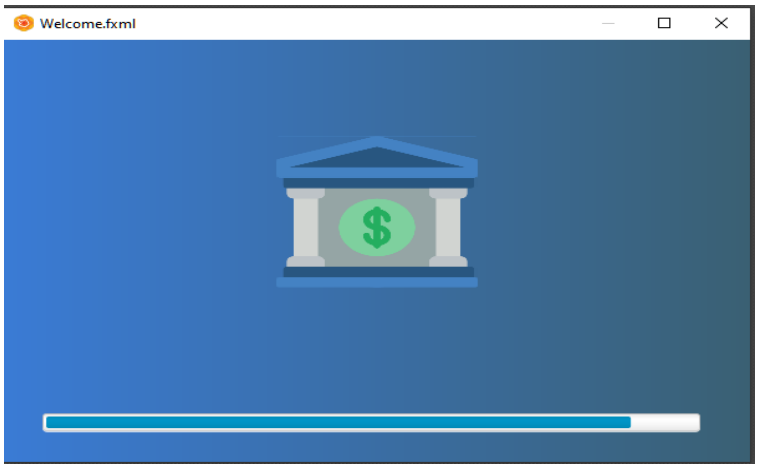
Alternativa	FACTIBLE	EFICIENTE	USO Y MANEJO	COMPLEJIDAD	INTEGRAL
1	4	5	3	4	5
2	3	4	1	3	3
3	4	4	2	5	4

Alternativa 1: 21 pts
Alternativa 2: 14 pts
Alternativa 3: 19 pts

Imágenes

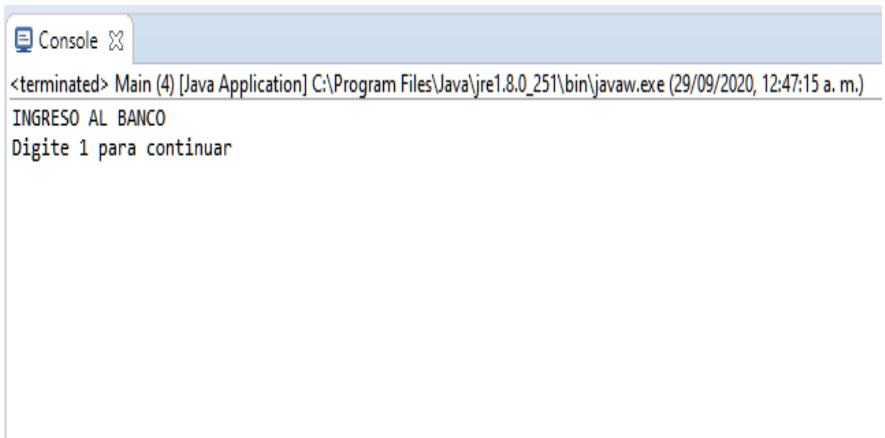
Se muestra una ventana de inicio la cual está ordenada respectivamente con la alternativa solución.

1.



2.

3.



Conclusión:

La solución escogida es la 1 por su respectivo puntaje, presentando un cumplimiento con todos los requerimientos, modelando una solución pertinente, la cual puede tener mejoras,pero cumple con el requerimiento de los ordenamiento,el algoritmo de complejidad O(n^2) que vamos a utilizar es el Selection Sort, luego entonces al escogimos los otros 3 algoritmos que nos ayudaran a resolver nuestro problema decidimos incluir los algoritmos de Heapsort, el Merge Sort, Quicksort.

En adición teniendo en cuenta que nuestro software es para un usuario, el diseño de la interfaces con un alto atractivo, genera una usabilidad, manejo del software más óptimo,

otro punto es la implementación propia de todos las estructuras de datos con generics, tales como hashtables, priority queue, stacks,queues, algoritmos de ordenamiento, y demás.

Siendo esta una solución integral y adecuada a la rúbrica de calificación.

Bibliografía

https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja

https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n

https://es.wikipedia.org/wiki/Ordenamiento_por_mezcla

https://es.wikipedia.org/wiki/Ordenamiento_por_cuentas

https://es.wikipedia.org/wiki/Quicksort

https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n

https://docs.oracle.com/javase/7/docs/api/

https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html

https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html

https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html

https://www.geeksforgeeks.org/

https://definicion.de/

https://drive.google.com/file/d/0Bwy19LIX4H2RXzFLUWZYY3N5cmM/view

https://pereiratechtalks.com/analisis-de-algoritmos-de-ordenamiento/

https://es.wikipedia.org/wiki/Eficiencia_algor%C3%ADtmica#Complejidad_espacial

https://es.wikipedia.org/wiki/Complejidad_temporal#:~:text=En%20inform%C3%A1tica%2C%20la%20complejidad%20temporal,que%20lleva%20ejecutar%20un%20algoritmo.&text=Por%20lo%20tanto%2C%20la%20cantidad,un%20factor%20constante%20como%20m%C3%A1ximo.