UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ALUMNO:

PAZ MALDONADO CARLOS SAÚL

NOMBRE DEL PROFESOR:

HERNANDEZ CABRERA JESUS

NOMBRE DE LA MATERIA:

ESTRUCTURA DE DATOS

FECHA DE ENTREGA:

19 de septiembre del 2024

TAREA NO. 8

1. Instrucciones de la tarea.

Criterios de evaluación.

- Debe emplear la clase ColaConPrioridadAcotada.java desarrollada en clase o equivalente en el lenguaje seleccionado.
- Debe diseñar e implementar la clase ClienteBanco.
- Debe contar con una clase con un main() en donde:

1. Lleguen 2 clientes nuevos.
2. Lleguen 3 personas que no son clientes.
3. Llega una celebridad.
4. Imprime el estado de la cola con prioridad acotada.
5. Se atienda al siguiente cliente, en donde retire $10,000 de su cuenta
6. llegan dos clientes más, uno frecuente y un premium.
7. Atender al siguiente cliente
8. Imprime el estado de la cola con prioridad acotada.
9. Atender todos los clientes restantes.
10. Imprime el estado de la cola con prioridad acotada

NOTA:

Para crear la clase ColaConPrioridadAcotadaADT utilicé la herencia de ColaConPrioridadADT, que a su vez es heredada por ColaADT.

La base de la ColaADT son los nodos dobles con un atributo de prioridad.

2. Capturas de ejecución.

```
AGREGANDO 2 CLIENTES NUEVOS
AGREGANDO 3 PERSONAS QUE NO SON CLIENTES
AGREGANDO A CELEBRIDAD


IMPRIMIENDO ESTADO DE LA COLA ACTUAL:
[Nombre Edad Dinero Tipo]
[Carlos Paz 18 $20000.0 4] <--> [Armando Perez 20 $400.0 4] <--> [Alejandra Hernandez 20 $0.0 5] <--> [Ismael Castillo 20 $40.0 5] <--> [Ana Cervantes 20 $4000.0 5] <--> [Esmeralda Gutierrez 20 $100000.0 1]

ATENDIENDO SIGUIENTE CLIENTE, DONDE RETIRA $10,000:


LLEGAN DOS CLIENTES MAS
CLIENTE FRECUENTE
CLIENTE PREMIUM
ATENDIENDO SIGUIENTE CLIENTE:
Armando Perez 20 $400.0 4


IMPRIMIENDO ESTADO DE LA COLA ACTUAL:
[Nombre Edad Dinero Tipo]
[Alejandra Hernandez 20 $0.0 5] <--> [Ismael Castillo 20 $40.0 5] <--> [Ana Cervantes 20 $4000.0 5] <--> [Esmeralda Gutierrez 20 $100000.0 1] <--> [Sebastian Espinoza 30 $10000.0 3] <--> [Estefania Licea 22
```

```
ATENDIENDO EL SIGUIENTE CLIENTE:
Alejandra Hernandez 20 $0.0 5
ATENDIENDO EL SIGUIENTE CLIENTE:
Ismael Castillo 20 $40.0 5
ATENDIENDO EL SIGUIENTE CLIENTE:
Ana Cervantes 20 $4000.0 5
ATENDIENDO EL SIGUIENTE CLIENTE:
Esmeralda Gutierrez 20 $100000.0 1
ATENDIENDO EL SIGUIENTE CLIENTE:
Sebastian Espinoza 30 $10000.0 3
ATENDIENDO EL SIGUIENTE CLIENTE:
Estefania Licea 22 $10000.0 2


Estado de la cola actual:
 Tamaño: 0

Process finished with exit code 0
```

3. Capturas del código.

a. Clase **MAIN**

```
package unam.fesaragon.estructuradatos;

import unam.fesaragon.estructuradatos.adt.colaadtconprioridad.ColaConPrioridadADT;
import unam.fesaragon.estructuradatos.adt.colaadtconprioridad.ColaConPrioridadAcotadaADT;
import unam.fesaragon.estructuradatos.banco.BancoTiposDeCliente;
import unam.fesaragon.estructuradatos.banco.ClienteBanco;

public class Main {
    public static void main(String[] args) {
        ColaConPrioridadADT<ClienteBanco> colaAcotada = new ColaConPrioridadAcotadaADT<>(5);
        imp("\n\nAGREGANDO 2 CLIENTES NUEVOS ");
        colaAcotada.encolar(new ClienteBanco("Carlos", "Paz", 18, 203423003, 20000f,
BancoTiposDeCliente.CLIENTE_NUEVO));
        colaAcotada.encolar(new ClienteBanco("Armando", "Perez", 20, 203423004, 400f,
BancoTiposDeCliente.CLIENTE_NUEVO));

        imp("AGREGANDO 3 PERSONAS QUE NO SON CLIENTES ");
        colaAcotada.encolar(new ClienteBanco("Alejandra", "Hernandez", 20, 203423005, 0f,
BancoTiposDeCliente.NO_ES_CLIENTE));
        colaAcotada.encolar(new ClienteBanco("Ismael", "Castillo", 20, 203423006, 40f,
BancoTiposDeCliente.NO_ES_CLIENTE));
        colaAcotada.encolar(new ClienteBanco("Ana", "Cervantes", 20, 203423007, 4000f,
BancoTiposDeCliente.NO_ES_CLIENTE));

        imp("AGREGANDO A CELEBRIDAD ");
        colaAcotada.encolar(new ClienteBanco("Esmeralda", "Gutierrez", 20, 203423008, 100000f,
BancoTiposDeCliente.CELEBRIDADES));

        imp("\n\nIMPRIMIENDO ESTADO DE LA COLA ACTUAL: ");
        imp("[Nombre Edad Dinero Tipo]");
        imp(colaAcotada.toString());

        imp("\n\nATENDIENDO SIGUIENTE CLIENTE, DONDE RETIRA $10,000: ");
        colaAcotada.desEncolar().retirarDinero(10000f);

        imp("\n\nLLEGAN DOS CLIENTES MAS ");
        imp("CLIENTE FRECUENTE ");
        colaAcotada.encolar(new ClienteBanco("Sebastian", "Espinoza", 30, 203423009, 10000f,
BancoTiposDeCliente.CLIENTE_FRECUENTE));

        imp("CLIENTE PREMIUM ");
        colaAcotada.encolar(new ClienteBanco("Estefania", "Licea", 22, 203423010, 10000f,
BancoTiposDeCliente.CLIENTE_PREMIUM));

        imp("ATENDIENDO SIGUIENTE CLIENTE: ");
        imp(colaAcotada.desEncolar().toString());

        imp("\n\nIMPRIMIENDO ESTADO DE LA COLA ACTUAL: ");
        imp("[Nombre Edad Dinero Tipo]");
        imp(colaAcotada.toString());
        imp("\n\n");
        while (!colaAcotada.estaVacia()) {
            imp("ATENDIENDO EL SIGUIENTE CLIENTE: ");
            imp(colaAcotada.desEncolar().toString());
        }

        imp("\n\nEstado de la cola actual:\n" + colaAcotada.toString());
    }

    private static void imp(String texto) {
        System.out.println(texto);
    }
}
```

b. Clase **ColaConPrioridadADT**

```java
package unam.fesaragon.estructuradatos.adt.colaadtconprioridad;

import unam.fesaragon.estructuradatos.adt.colaadt.ColaADT;

public class ColaConPrioridadADT<T> extends ColaADT<T> {

    protected ListaDoblementeLigadaConPrioridad<T> datos;

    //CONSTRUCTOR
    public ColaConPrioridadADT() {
        this.datos = new ListaDoblementeLigadaConPrioridad<>();
        setData(this.datos);
    }

    //METODOS

    @Override
    public int longitud() {
        return this.datos.getTamanio();
    }

    public void encolar(int prioridad, T valor) {
        // Algoritmo para encolar según su prioridad
        datos.agregarAlFinal(valor, prioridad);
    }


    @Override
    public T desEncolar() {
        //Obtener el primer elemento
        T dato = this.datos.obtener(1);
        //Despues eliminarlo
        this.datos.eliminar_el_primero();
        return dato;

    }

    @Override
    public String toString() {
        return datos.toString();
    }
}
```

c. Clase **ColaConPriordadAcotadaADT**

```
package unam.fesaragon.estructuradatos.adt.colaadtconprioridad;

public class ColaConPrioridadAcotadaADT<T> extends ColaConPrioridadADT<T> {
    private int prioridadAcotada;

    public ColaConPrioridadAcotadaADT(int prioridadAcotadaMaxima) {
        super();
        this.prioridadAcotada = prioridadAcotadaMaxima;
    }

    @Override
    public void encolar(int prioridad, T valor) {
        if (prioridad > 0 && prioridad <= prioridadAcotada) {
            super.encolar(prioridad, valor);
        } else {
            System.out.println("No se puede ingresar por que la prioridad esta fuera
del rango acotado: " + this.prioridadAcotada);
        }
    }

}
```

d. Clase **ListaDoblementeLigadaConPrioridad**

```
package unam.fesaragon.estructuradatos.adt.colaadtconprioridad;

import unam.fesaragon.estructuradatos.adt.colaadt.ListaDoblementeLigada;

class ListaDoblementeLigadaConPrioridad<T> extends ListaDoblementeLigada<T> {

    public ListaDoblementeLigadaConPrioridad() {
        super();
    }

    public ListaDoblementeLigadaConPrioridad(NodoDobleCola<T> head, NodoDobleCola<T>
tail) {
        super(head, tail);
    }

    public void agregarAlFinal(T valor, int prioridad) {
        NodoDobleCola<T> nodoAInsertar = new NodoDobleCola<>(valor, prioridad);

        if (this.estaVacia()) {
            this.setHead(nodoAInsertar);
            this.setTail(nodoAInsertar);
        } else {
            NodoDobleCola<T> nodoReferencia = obtenerNodoReferencia(prioridad);

            if (nodoReferencia == null) {
                insertarDespues((NodoDobleCola<T>) this.getTail(), nodoAInsertar);
            } else if (nodoReferencia.getPrioridad() < prioridad) {
                insertarDespues(nodoReferencia, nodoAInsertar);
```

```
            } else if (nodoReferencia.getPrioridad() == prioridad) {
                insertarDespues(obtUltimoNodoMismaPri(prioridad), nodoAInsertar);
            } else {
                insertarAntes(nodoReferencia, nodoAInsertar);
            }
        }

        this.setTamanio(this.getTamanio() + 1);
    }

    private NodoDobleCola<T> obtenerNodoReferencia(int prioridad) {
        if (exMismaPrioridad(prioridad)) {
            return obtUltimoNodoMismaPri(prioridad);
        } else if (exPrioAnterior(prioridad)) {
            return obtNodoPrioridadAnterior(prioridad);
        } else {
            return (NodoDobleCola<T>) this.getHead();
        }
    }

    private void insertarDespues(NodoDobleCola<T> nodoReferencia, NodoDobleCola<T>
nodoAInsertar) {
        nodoAInsertar.setAnterior(nodoReferencia);
        nodoAInsertar.setSiguiente(nodoReferencia.getSiguiente());

        if (nodoReferencia.getSiguiente() != null) {
            nodoReferencia.getSiguiente().setAnterior(nodoAInsertar);
        }

        nodoReferencia.setSiguiente(nodoAInsertar);

        if (nodoReferencia == this.getTail()) {
            this.setTail(nodoAInsertar);
        }
    }

    private void insertarAntes(NodoDobleCola<T> nodoReferencia, NodoDobleCola<T>
nodoAInsertar) {
        nodoAInsertar.setSiguiente(nodoReferencia);
        nodoAInsertar.setAnterior(nodoReferencia.getAnterior());

        if (nodoReferencia.getAnterior() != null) {
            nodoReferencia.getAnterior().setSiguiente(nodoAInsertar);
        }

        nodoReferencia.setAnterior(nodoAInsertar);

        if (nodoReferencia == this.getHead()) {
            this.setHead(nodoAInsertar);
        }
    }

    private boolean exMismaPrioridad(int prioridad) {
```

```java
        NodoDobleCola<T> aux = (NodoDobleCola<T>) this.getHead();
        while (aux != null) {
            if (aux.getPrioridad() == prioridad) {
                return true;
            }
            aux = (NodoDobleCola<T>) aux.getSiguiente();
        }
        return false;
    }

    private boolean exPrioAnterior(int prioridad) {
        NodoDobleCola<T> aux = (NodoDobleCola<T>) this.getTail();
        while (aux.getAnterior() != null) {
            if (aux.getPrioridad() < prioridad) {
                return true;
            }
            aux = (NodoDobleCola<T>) aux.getAnterior();
        }
        return false;
    }

    private NodoDobleCola<T> obtUltimoNodoMismaPri(int prioridad) {
        NodoDobleCola<T> aux = (NodoDobleCola<T>) this.getHead();
        NodoDobleCola<T> ultimoConMismaPrioridad = null;

        while (aux != null) {
            if (aux.getPrioridad() == prioridad) {
                ultimoConMismaPrioridad = aux;
            }
            aux = (NodoDobleCola<T>) aux.getSiguiente();
        }

        return ultimoConMismaPrioridad;
    }

    private NodoDobleCola<T> obtNodoPrioridadAnterior(int prioridad) {
        NodoDobleCola<T> aux = (NodoDobleCola<T>) this.getTail();
        while (aux.getAnterior() != null) {
            if (aux.getPrioridad() < prioridad) {
                return aux;
            }
            aux = (NodoDobleCola<T>) aux.getAnterior();
        }
        return aux;
    }
}
```

e. Clase **NodoDobleCola**

```java
package unam.fesaragon.estructuradatos.adt.colaadtconprioridad;

import unam.fesaragon.estructuradatos.adt.colaadt.NodoDoble;

class NodoDobleCola<T> extends NodoDoble<T> {
    private int prioridad;

    public NodoDobleCola() {
    }

    public NodoDobleCola(T dato, int prioridad) {
        super(dato);
        this.prioridad = prioridad;
    }

    public NodoDobleCola(T dato, NodoDoble<T> anterior, NodoDoble<T> siguiente, int
prioridad) {
        super(dato, anterior, siguiente);
        this.prioridad = prioridad;
    }

    public int getPrioridad() {
        return prioridad;
    }

    public void setPrioridad(int prioridad) {
        this.prioridad = prioridad;
    }
}
```

f. Clase **ColaADT**

```java
package unam.fesaragon.estructuradatos.adt.colaadt;

public class ColaADT<T> {
    private ListaDoblementeLigada<T> data;

    //CONSTRUTOR
    public ColaADT() {
        this.data = new ListaDoblementeLigada<>();
    }

    //METODOS
    public boolean estaVacia() {
        return this.data.estaVacia();
    }

    public int longitud() {
        return this.data.getTamanio();
    }
```

```
    public T frente() {
        return this.data.obtener(1);
    }

    public void encolar(T valor) { //enqueue
        this.data.agregarAlFinal(valor);
    }

    public T desEncolar() {
        //Obtener el primer elemento
        T dato = this.data.obtener(1);
        //Despues eliminarlo
        this.data.eliminar_el_primero();
        return dato;
    }

    public T siguiente() {
        return this.data.obtener(2);
    }

    public void setData(ListaDoblementeLigada<T> data) {
        this.data = data;
    }

    @Override
    public String toString() {
        return this.data.toString();
    }
}
```

g. Clase ListaDoblementeLigada

```
package unam.fesaragon.estructuradatos.adt.colaadt;

public class ListaDoblementeLigada<T> {
    private NodoDoble<T> head;
    private NodoDoble<T> tail;
    private int tamanio;

    // Constructor
    public ListaDoblementeLigada() {
        this.head = null;
        this.tail = null;
        this.tamanio = 0;
    }

    public ListaDoblementeLigada(NodoDoble<T> head, NodoDoble<T> tail) {
        this.head = head;
        this.tail = tail;
    }

    // Comprobar si está vacía
```

```java
    public boolean estaVacia() {
        return this.tamanio == 0;
    }

    // Agregar al inicio de la lista
    public void agregar_al_inicio(T valor) {
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        if (estaVacia()) {
            this.head = nuevo;
            this.tail = nuevo;
        } else {
            nuevo.setSiguiente(this.head);
            this.head.setAnterior(nuevo);
            this.head = nuevo;
        }
        tamanio++;
    }

    // Agregar al final de la lista
    public void agregarAlFinal(T valor) {
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        if (estaVacia()) {
            this.head = nuevo;
            this.tail = nuevo;
        } else {
            this.tail.setSiguiente(nuevo);
            nuevo.setAnterior(this.tail);
            this.tail = nuevo;
        }
        tamanio++;
    }

    // Agregar después de un nodo de referencia
    public void agregar_después_de(T referencia, T valor) {
        if (estaVacia()) {
            System.out.println("La lista esta vacia y por lo tanto no existe la
referencia");
            return;
        }

        NodoDoble<T> aux = this.head;
        while (aux != null) {
            if (aux.getDato().equals(referencia)) {
                break;
            }
            aux = aux.getSiguiente();
        }

        if (aux == null) {
            System.out.println("El nodo de referencia no existe");
            return;
        }
```

```
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        nuevo.setSiguiente(aux.getSiguiente());
        nuevo.setAnterior(aux);
        //Nodo de referencia, no es la ultima en la lista.
        if (aux.getSiguiente() != null) {
            aux.getSiguiente().setAnterior(nuevo);
        } else {
            this.tail = nuevo;
        }
        aux.setSiguiente(nuevo);
        this.tamanio++;
    }

    // Obtener el elemento en una posición específica
    public T obtener(int posicion) {
        //Condiciones
        if (estaVacia()) {
            System.out.println("La lista está vacía");
            return null;
        }
        if (posicion > this.tamanio) {
            System.out.println("Posición fuera de rango");
            return null;
        }

        NodoDoble<T> aux = this.head;
        for (int i = 1; i < posicion; i++) {
            aux = aux.getSiguiente();
        }

        return aux.getDato();
    }

    // Eliminar el primer elemento
    public void eliminar_el_primero() {
        if (estaVacia()) {
            System.out.println("La lista está vacía");
            return;
        }
        if (this.head == this.tail) { // Solo un elemento
            this.head = null;
            this.tail = null;
        } else {
            this.head = this.head.getSiguiente();
            this.head.setAnterior(null);
        }
        tamanio--;
    }

    // Eliminar el último elemento
    public void eliminar_el_final() {
        if (estaVacia()) {
            System.out.println("La lista está vacía");
```

```
            return;
        }

        if (this.head == this.tail) { // Solo un elemento
            this.head = null;
            this.tail = null;
        } else {
            this.tail = this.tail.getAnterior();
            this.tail.setSiguiente(null);
        }
        tamanio--;
    }

    // Eliminar un elemento en una posición específica
    public void eliminar(int posicion) {
        if (estaVacia()) {
            System.out.println("La lista está vacía.");
            return;
        }
        if (posicion > this.tamanio) {
            System.out.println("Posición fuera de rango");
            return;
        }

        if (posicion == 1) {
            eliminar_el_primero();
            return;
        }
        if (posicion == this.tamanio) {
            eliminar_el_final();
            return;
        }

        NodoDoble<T> aux = this.head;
        int i = 1;
        while (i < posicion) {
            aux = aux.getSiguiente();
            i++;
        }

        aux.getAnterior().setSiguiente(aux.getSiguiente());
        aux.getSiguiente().setAnterior(aux.getAnterior());
        this.tamanio--;
    }

    // Buscar un elemento y retornar su posición
    public int buscar(T valor) {
        if (estaVacia()) {
            System.out.println("La lista está vacía");
            return 0;
        }

        NodoDoble<T> aux = this.head;
```

```
        int posicion = 1;
        while (aux != null) {
            if (aux.getDato().equals(valor)) {
                return posicion;
            }
            aux = aux.getSiguiente();
            posicion++;
        }
        System.out.println("No se encontro el elemento");
        return 0;
    }

    // Actualizar un elemento
    public void actualizar(T a_buscar, T valor) {
        if (estaVacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        NodoDoble<T> aux = this.head;
        while (aux != null) {
            if (aux.getDato().equals(a_buscar)) {
                aux.setDato(valor);
                return;
            }
            aux = aux.getSiguiente();
        }
        System.out.println("El elemento valor o elemento no esta en la lista");
    }

    // Recorrido transversal en una dirección específica
    // Verdadero es valor por defecto y con el falso es de derecha a izquierda
    public void transversal(boolean porDefecto) {
        if (estaVacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        if (porDefecto) {
            NodoDoble<T> aux = this.head;
            while (aux != null) {
                System.out.print("[" + aux.getDato() + "] <--> ");
                aux = aux.getSiguiente();
            }
        } else {
            NodoDoble<T> aux = this.tail;
            while (aux != null) {
                System.out.print("[" + aux.getDato() + "] <--> ");
                aux = aux.getAnterior();
            }
        }
        System.out.println("NULL");
    }
```

```java
    //Sobrecargue el metodo para poner por default el recorrido de izquierda a derecha
    public void transversal() {
        transversal(true);
    }

    //GETTERS Y SETTERS
    // Obtener el tamaño de la lista
    public int getTamanio() {
        return this.tamanio;
    }

    public void setTamanio(int tamanio) {
        this.tamanio = tamanio;
    }

    public NodoDoble<T> getHead() {
        return head;
    }

    public void setHead(NodoDoble<T> head) {
        this.head = head;
    }

    public NodoDoble<T> getTail() {
        return tail;
    }

    public void setTail(NodoDoble<T> tail) {
        this.tail = tail;
    }

    //ToString
    @Override
    public String toString() {
        StringBuilder estado = new StringBuilder();
        NodoDoble<T> aux = this.head;
        while (aux != null) {
            estado.append("[").append(aux.getDato()).append("] <--> ");
            aux = aux.getSiguiente();
        }
        return estado + " Tamaño: " + this.tamanio;
    }
}
```

h. Clase **NodoDoble**

```java
package unam.fesaragon.estructuradatos.adt.colaadt;

public class NodoDoble<T> {
    private T dato;
    private NodoDoble<T> anterior;
    private NodoDoble<T> siguiente;

    // Constructores
    public NodoDoble() {
    }

    public NodoDoble(T dato) {
        this.dato = dato;
        this.anterior = null;
        this.siguiente = null;
    }

    public NodoDoble(T dato, NodoDoble<T> anterior, NodoDoble<T> siguiente) {
        this.dato = dato;
        this.anterior = anterior;
        this.siguiente = siguiente;
    }

    // Getters y Setters
    public T getDato() {
        return dato;
    }

    public void setDato(T dato) {
        this.dato = dato;
    }

    public NodoDoble<T> getAnterior() {
        return anterior;
    }

    public void setAnterior(NodoDoble<T> anterior) {
        this.anterior = anterior;
    }

    public NodoDoble<T> getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoDoble<T> siguiente) {
        this.siguiente = siguiente;
    }
}
```