1. Functional requirements

Name:	R. #1. Assign turns at the time of admission
Summary:	Assign turns to users with name and ID data
Input:	String name, int id
Results:	Turn assigned to user
	Incomplete data, turn not assigned

Name:	R. #2. Place a user in the queue
Summary:	Place a user in the queue. There are two queue, the customer and the priority
Input:	
Results:	User located in queue

Name:	R. #3. Search a user in data base
Summary:	Search efficiently a user in data base to get information about user before it reaches the office where it will be attended
Input:	int id
Results:	User was found
	User wasn't found

Name:	R. #4. Change the amount of the customer’s saving account
Summary:	Change the amount of the customer’s saving account when requesting a withdrawal or consignment
Input:	int id
Results:	Changed the amount of the customer
	User does not have a saving account

Name:	R. #5. Cancellation of a customer’s account
Summary:	Delate client data from database and add it to an exclusive datbase for those who desert
Input:	
Results:	Changed the amount of the customer
	User does not have a saving account

Name:	R. #6. Card payment
Summary:	User can pay amount used by credit card until now. Payment can be by cash or saving account
Input:	int id
Results:	Paid card
	User does not have debts

Name:	R. #7. Undo functionality
Summary:	Undo functionality works to fix mistakes, even after save those
Input:	int id
Results:	Paid card
	User does not have debts

2. Non-functional requirements

- Implement a ordering method for each of the chosen parameters, with the restriction that only one of them can have averge temporal complexity of n^2
- Users information must be displayed in a spreadsheet-type table
- Must have a graphical interface to display users information and make operations, spreadsheet-type table and status of queues

3. Test documentation

Nombre	Clase	Escenario
setUp1	CashierModuleTest	Un objeto de la clase CashierModule sin clientes
setUp2	CashierModuleTest	Un objeto de la clase CashierModule con 4 clientes con atributos: name = “Andrea” id = 001 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020” name = “Danna” id = 257 genero = true condición especial = 2 monto = 2 cupo = 3 fecha = “29/09/2020” name = “Camilo” id = 999 genero = false condición especial = 0 monto = 10000 cupo = 0 fecha = “29/09/2020” name = “Aristizabal” id = 666 genero = true condición especial = 0 monto = 0 cupo = 0 fecha = “29/09/2020”

Objetivo de la prueba: Determinar si el método signUpClient agrega correctamente a un cliente nuevo

Clase	Método	Escenario	Valores de Entrada	Resultado
CashierModuleTest	**signUpClientTest**	signUp1	4 objetos cliente con los atributos name = “Andrea” id = 001 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020” name = “Danna” id = 257 genero = true condición especial = 2 monto = 2 cupo = 3 fecha = “29/09/2020” name = “Camilo” id = 999 genero = false condición especial = 0 monto = 10000 cupo = 0 fecha = “29/09/2020” name = “Aristizabal” id = 666 genero = true condición especial = 0 monto = 0 cupo = 0 fecha = “29/09/2020”	**Los clientes se agregan correctamente**

Objetivo de la prueba: Determinar si el método unifyClient unifica correctamente y de manera ordenada los clientes de las diferentes particiones de la base de datos en el ArrayList				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**

CashierModuleTest	**unifyClientTest**	signUp2	4 objetos cliente con los atributos name = “Andrea” id = 001 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020” name = “Danna” id = 257 genero = true condición especial = 2 monto = 2 cupo = 3 fecha = “29/09/2020” name = “Camilo” id = 999 genero = false condición especial = 0 monto = 10000 cupo = 0 fecha = “29/09/2020” name = “Aristizabal” id = 666 genero = true condición especial = 0 monto = 0 cupo = 0 fecha = “29/09/2020” Un objeto ArrayList de tipo cliente sin ningún cliente	**Los clientes se unifican correctamente y de forma ordenada en el ArrayList**

Objetivo de la prueba: Determinar si el método search busca correctamente a un cliente

Clase	Método	Escenario	Valores de Entrada	Resultado
CashierModuleTest	**searchTest**	signUp2		**El cliente se busca de manera correcta y retorna el cliente que se buscaba**

Objetivo de la prueba: Determinar si el método deleteClientAccount elimina correctamente a un cliente				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
CashierModuleTest	**deleteClientAccountTest**	signUp2		**El cliente se elimina correctamente y se elimina el cliente que se necesitaba eliminar**

Nombre	Clase	Escenario
setUp1	QueueModuleTest	Un objeto de la clase QueueModuleTest sin usuarios

setUp2	QueueModuleTest	Un objeto de la clase QueueModuleTest con 3 usuarios con atributos: name = “Andrea” id = 001 genero = true condición especial = 0 name = “Danna” id = 257 genero = true condición especial = 2 name = “Camilo” id = 999 genero = false condición especial = 0

Objetivo de la prueba: Determinar si el método receivePerson recibe correctamente un usuario y lo asigna en la fila que le corresponde dependiendo de su condición especial

Clase	Método	Escenario	Valores de Entrada	Resultado
QueueModuleTest	**receivePersonTest**	setUp1	2 objetosde la clase usuario con los atributos name = “Andrea” id = 001 genero = true condición especial = 0 name = “Danna” id = 257 genero = true condición especial = 2	**Los usuarios se reciben correctamente en las filas que le corresponden dependiendo de su condición especial**

Nombre	Clase	Escenario
setUp1	QueueTest	Un objeto de la clase QueueTest con un solo usuario con los atributos name = “Danna” id = 257 genero = true condición especial = 2
setUp2	QueueTest	Un objeto de la clase QueueTest con 2 usuarios con atributos: name = “Andrea” id = 001 genero = true condición especial = 0 name = “Danna” id = 257 genero = true condición especial = 2

Objetivo de la prueba: Determinar si el método dequeueTestLimits desencola correctamente un usuario en el caso límite de que la cola no tenga ningún usuario

Clase	Método	Escenario	Valores de Entrada	Resultado

queueTest	**dequeueTestLimits**	setUp1		**El método controla correctamente la Excepcion cuando se intenta desencolar un elemento que es null**

Objetivo de la prueba: Determinar si el método dequeue desencola correctamente un usuario

Clase	Método	Escenario	Valores de Entrada	Resultado
QueueTest	**dequeueTestCaseBase**	setUp2		**El método desencola correctamente el usuario que se le solicita**

Objetivo de la prueba: Determinar si el método enqueue atrapa correctamente la Excepcion al intentar encolar un elemento nulo

Clase	Método	Escenario	Valores de Entrada	Resultado
QueueTest	**enqueueTestInterestingCase**	setUp1		**El método controla correctamente la Excepcion**

Objetivo de la prueba: Determinar si el método enqueue verifica que el último método encolado sea el último elemento en la fila correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
QueueTest	**enqueueLastTest**	setUp1	Objeto usuario con los atributos name = “Camilo” id = 999 genero = false condición especial = 0	**El método controla correctamente la Excepcion**

Objetivo de la prueba: Determinar si el método isEmpty retorna correctamente si la cola está vacía

Clase	Método	Escenario	Valores de Entrada	Resultado
QueueTest	**isEmptyTest**	setUp1		**El método determina correctamente que la cola esta vacía**

Objetivo de la prueba: Determinar si el método size retorna correctamente el tamaño de la cola

Clase	Método	Escenario	Valores de Entrada	Resultado

QueueTest	**sizeTest**	setUp1		**El método retorna correctamente el tamaño de la cola**

Nombre	Clase	Escenario
setUp1	HashTableTest	Un objeto de la clase MyHashTable sin clientes
setUp2	HashTableTest	Un objeto de la clase MyHashTable con 1 clientes con atributos: name = “Camilo” id = 999 genero = false condición especial = 0 monto = 10000 cupo = 0 fecha = “29/09/2020”

setUp3	HashTableTest	Un objeto de la clase MyHashTable con 2 clientes con atributos: name = “Camilo” id = 999 genero = false condición especial = 0 monto = 10000 cupo = 0 fecha = “29/09/2020” name = “Danna” id = 257 genero = true condición especial = 2 monto = 2 cupo = 3 fecha = “29/09/2020” name = “Andrea” id = 001 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020”

Objetivo de la prueba: Determinar si el método isEmpty retorna correctamente si la hash table está vacía				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**

HashTableTest	**isEmptyTest**	setUp1		**El método determina correctamente que la hash table esta vacía**
HashTableTest	**isEmptyTest2**	setUp2		**El método determina correctamente que la hash table no esta vacía**

Objetivo de la prueba: Determinar si el método get retorna correctamente el elemento en la posición solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
HashTableTest	**getTest**	setUp3		**El método retorna correctamente el cliente que se le solicita**
HashTableTest	**getTestInterestingCase**	setUp1		**El método retorna null ante un elemento que no existe dado que la hash table se encuentra vacia**

Objetivo de la prueba: Determinar si el método remove elimina correctamente el elemento solicitado en la hash table

Clase	Método	Escenario	Valores de Entrada	Resultado

HashTableTest	**removeTestSpecialCase**	setUp1		**El método determina correctamente si al eliminar en una hash table vacía retorna null**
HashTableTest	**removeTestBaseCase**	setUp2		**El método elimina correctamente el cliente que se solicita de la tabla hash**

Objet ivo de la prueba: Determinar si el método add agrega correctamente los clientes				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
HashTableTest	**addTest**	setUp3	Un objeto de la clase cliente con atributos: name = “Alberto” id = 111 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020”	**El método agrega correctamente el cliente que se solicita**
HashTableTest	**addTestInterestingCase**	setUp1	Un objeto de la clase cliente con atributos: name = “Alberto” id = 111 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020”	**El método agrega correctamente el cliente que se solicita**

Nombre	Clase	Escenario
setUp	MyDoublyLinkedListTester	Tres objetos de la clase MyDoublyLinkedListTester con 3 clientes con los siguientes atributos: name = “Andrea” id = 001 genero = true condición especial = 0 monto = 322 cupo = 0 fecha = “29/09/2020” name = “Danna” id = 257 genero = true condición especial = 2 monto = 2 cupo = 3 fecha = “29/09/2020” name = “Camilo” id = 999 genero = false condición especial = 0 monto = 10000 cupo = 0 fecha = “29/09/2020”

Objetivo de la prueba: Determinar si el método get retorna correctamente el elemento que se le pide

Clase	Método	Escenario	Valores de Entrada	Resultado
MyDoublyLinkedListTester	**testGet**	setUp	Un cliente con los siguientes atributos: name = “Andrea” id = 100 genero = true condición especial = 0 monto = 100 cupo = 0 fecha = “29/09/2020”	**El método retorna correctamente el elemento que se le pide**

Objetivo de la prueba: Determinar si el método remove elimina correctamente el elemento que se le pide				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
MyDoublyLinkedListTester	**testRemove**	setUp		**El método elimina correctamente el elemento que se le pide**

Objetivo de la prueba: Determinar si el método size retorna correctamente el tamaño de la lista				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
MyDoublyLinkedListTester	**testSize**	setUp		**El método retorna correctamente el tamaño de la lista**

Nombre	Clase	Escenario
setUp1	PriorityQueueHeapTest	Un objeto de la clase MyPriorityQueueHeap sin usuarios
setUp2	PriorityQueueHeapTest	Un objeto de la clase MyPriorityQueueHeap con 4 usuarios con atributos: name = “Andrea” id = 100 genero = true condición especial = 0 name = “Danna” id = 2 genero = true condición especial = 2 name = “Camilo” id = 4 genero = false condición especial = 2 name = “Cristhian” id = 6 genero = false condición especial = 1

setUp3	PriorityQueueHeapTest	Un objeto de la clase MyPriorityQueueHeap con 3 usuarios con atributos: name = “Andrea” id = 100 genero = true condición especial = 0 name = “Danna” id = 2 genero = true condición especial = 2 name = “Camilo” id = 4 genero = false condición especial = 2

Objetivo de la prueba: Determinar si el método enqueue agrega correctamente los usuarios				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
PriorityQueueHeapTest	**enqueueTest1**	setUp1	Objeto de la clase Usuario con los atributos: name = “Andrea” id = 100 genero = true condición especial = 0	**El método agrega correctamente el usuario que se solicita**

PriorityQueueHeapTest	**enqueueTest2**	setUp2	Objeto de la clase usuario con los atributos name = “Pinina” id = 4 genero = false condición especial = 3 name = “Andrea” id = 100 genero = true condición especial = 0	**El método agrega correctamente el cliente que se solicita**

Objetivo de la prueba: Determinar si el método dequeue agrega correctamente los usuarios				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
PriorityQueueHeapTest	**dequeueTest1**	setUp1		**El método retorna correctamente null al solicitarse desencolar en una fila vacía**

PriorityQueueHeapTest	**dequeueTest2**	setUp3	Un objeto de la clase usuario con los siguientes atributos name = “Camilo” id = 4 genero = false condición especial = 2	**El método desencola correctamente el usuario que se solicita**

Nombre	Clase	Escenario
setUp1	MyBSTTest	Un objeto de la clase MyBST sin usuarios
setUp2	PriorityQueueHeapTest	Un objeto de la clase MyBST con 3 usuarios con atributos: name = “Andrea” id = 100 genero = true condición especial = 0 name = “Danna” id = 2 genero = true condición especial = 2 name = “Camilo” id = 4 genero = false condición especial = 2

Objetivo de la prueba: Determinar si el método addNode agrega correctamente los clientes				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**

MyBSTTest	**addNodeTest1**	setUp1	Tres objetos de la clase Usuario con los siguientes atributos: name = “Andrea” id = 100 genero = true condición especial = 0 name = “Danna” id = 2 genero = true condición especial = 2 name = “Camilo” id = 4 genero = false condición especial = 2	**El método agrega correctamente los 3 usuarios en un árbol vacío**
MyBSTTest	**addNodeTest2**	setUp3	Un objeto de la clase usuario con los siguientes atributos: name = “Cristhian” id = 6 genero = false condición especial = 1	**El método agrega correctamente el usuario que se solicita**

Objetivo de la prueba: Determinar si el método removeNode elimina correctamente el usuario solicitado

Clase	Método	Escenario	Valores de Entrada	Resultado
MyBSTTest	**removeNodeTest**	setUp1	Un objeto de la clase usuario con los siguientes atributos name = “Andrea” id = 100 genero = true condición especial = 0	**El método elimina correctamente el usuario solicitado**

Objetivo de la prueba: Determinar si el método searchNode agrega correctamente los clientes

Clase	Método	Escenario	Valores de Entrada	Resultado
MyBSTTest	**searchNodeTest**	setUp1	Un objeto de la clase Usuario con los siguientes atributos: name = “Andrea” id = 100 genero = true condición especial = 0	**El método no encuentra el elemento porque es un árbol vacío, por lo tanto, está correcto**
MyBSTTest	**searchNodeTest1**	setUp1	Un objeto de la clase Usuario con los siguientes atributos: name = “Andrea” id = 100 genero = true condición especial = 0	**El busca correctamente el usuario solicitado**

Objetivo de la prueba: Determinar si el método isEmpty agrega correctamente los clientes

Clase	Método	Escenario	Valores de Entrada	Resultado
MyBSTTest	**isEmptyTest**	setUp1		**El método retorna true dado que el árbol si se encuentra vacío**
MyBSTTest	**isEmptyTest2**	setUp2		**El método retorna false dado que el árbol no se encuentra vacío**

Nombre	Clase	Escenario

setUp1	HeapSortTest	Un objeto de la clase myHeapSort y un objeto de la clase myGenericArray con 3 usuarios name = “Andrea” id = 100 genero = true condición especial = 0 name = “Danna” id = 2 genero = true condición especial = 2 name = “Camilo” id = 4 genero = false condición especial = 2
setUp2	HeapSortTest	Un objeto de la clase myHeapSort y un objeto de la clase myGenericArray sin usuarios
setUp3	HeapSortTest	Un objeto de la clase myHeapSort y un objeto de la clase myGenericArray con 9 usuarios name = “Andrea” id = 9 genero = true condición especial = 0 name = “Danna” id = 8 genero = true condición especial = 2 name = “Camilo” id = 7 genero = false condición especial = 2 name = “Christhian” id = 6 genero = false condición especial = 2 name = “Pinina” id = 5 genero = false condición especial = 2

		name = “Snow” id = 4 genero = false condición especial = 2 name = “Max” id = 3 genero = false condición especial = 2 name = “Lucy” id = 2 genero = false condición especial = 2

Objetivo de la prueba: Determinar si el método sort ordena correctamente los clientes

Clase	Método	Escenario	Valores de Entrada	Resultado
HeapSortTest	**sortTest1**	setUp1		**El método ordena correctamente los clientes**
HeapSortTest	**sortTest2**	setUp2		**El método ordena correctamente los clientes**
HeapSortTest	**sortTest3**	setUp3		**El método ordena correctamente los clientes**
HeapSortTest	**sortTest4**	setUp3		**El método ordena correctamente los clientes**