**Nomaan Khan**

**CS 4348-001**

**Project 2**

**Design**

**A list of every semaphore, its purpose, and its initial value:**

1. **Public Static Semaphore guestMutex;**

Mutex for guest thread queue operations.

Initial value: (1, true)

1. **Public Static Semaphore guestAvailable;**

Semaphore for when guest is ready to be dequeued.

Initial value: (0, true)

1. **Public Static Semaphore bellhopRequested;**

Semaphore for when guest requests a bellhop.

Initial value: (0, true)

1. **Public Static Semaphore frontDeskMutex;**

Mutex for front desk queue operations.

Initial value: (1, true)

1. **Public Static Semaphore frontDeskEmp;**

Semaphore for front desk employees.

Initial value: (2, true)

1. **Public Static Semaphore leftFrontDesk;**

Semaphore for when guest leaves front desk.

Initial value: (0, true)

1. **Public Static Semaphore bellhopMutex;**

Mutex for bellhop queue operations.

Initial value: (1, true)

1. **Public Static Semaphore bellhop;**

Semaphore for bellhops.

Initial value: (2, true)

1. **Public Static Semaphore bagsDelivered;**

Semaphore for when bags are delivered by bellhop.

Initial value: (0, true)

1. **Public Static Semaphore roomAssigned;**

Semaphore for when room is assigned to guest.

Initial value: (0, true)

1. **Public Static Semaphore bagsReceived;**

Semaphore for when bags are received by guest.

Initial value: (0, true)

1. **Public Static Semaphore reachedRoom[];**

When guests have reached their rooms.

Initial value: (0, true)

1. **Public Static Semaphore guestsCompleted[];**

When guest threads are terminated.

Initial value: (0, true)

**Pseudocode for each function and Class:**

**Public Class Hotel implements Runnable**

Public Static Queue<Guest> guestQueue; // Queue for guests to wait for front desk employees.

Public Static Queue<Guest> bellhopQueue; // Queue for guests to wait for bellhops.

Public Static Semaphore guestMutex; // Mutex for guest thread queue operations.

Public Static Semaphore guestAvailable; // Semaphore for when guest is ready to be dequeued.

Public Static Semaphore bellhopRequested; // Semaphore for when guest requests a bellhop.

Public Static Semaphore frontDeskMutex; // Mutex for front desk queue operations.

Public Static Semaphore frontDeskEmp; // Semaphore for front desk employees.

Public Static Semaphore leftFrontDesk; // Semaphore for when guest leaves front desk.

Public Static Semaphore bellhopMutex; // Mutex for bellhop queue operations.

Public Static Semaphore bellhop; // Semaphore for bellhops.

Public Static Semaphore bagsDelivered; // Semaphore for when bags are delivered by bellhop.

Public Static Semaphore roomAssigned; // Semaphore for when room is assigned to guest.

Public Static Semaphore bagsReceived; // Semaphore for when bags are received by guest.

Public Static Semaphore reachedRoom[]; // When guests have reached their rooms.

Public Static Semaphore guestsCompleted[]; // When guest threads are terminated.

Public Static int guestJoined;

Public Static int frontDeskID[]; // Array to store IDs of front desk employees for each guest.

Public Static int bellhopID[]; // Array to store IDs of bellhops for each guest.

Public Thread hotel;

Public Constructor Function Hotel()

guestAvailable = new Semaphore(0, true);

bellhopRequested = new Semaphore(0, true);

leftFrontDesk = new Semaphore(0, true);

bagsDelivered = new Semaphore(0, true);

bagsReceived = new Semaphore(0, true);

guestMutex = new Semaphore(1, true);

frontDeskMutex = new Semaphore(1, true);

bellhopMutex = new Semaphore(1, true);

roomAssigned = new Semaphore(0, true);

frontDeskEmp = new Semaphore(2, true);

bellhop = new Semaphore(2, true);

guestsCompleted = new Semaphore[25];

reachedRoom = new Semaphore[25];

guestQueue = new LinkedList<Guest>();

bellhopQueue = new LinkedList<Guest>();

frontDeskID = new int [25];

bellhopID = new int [25];

guestJoined = 0;

for int i = 0 until i < 25 do i++

guestsCompleted[i] = new Semaphore(0, true);

reachedRoom[i] = new Semaphore(0, true);

frontDeskID[i] = 0;

bellhopID[i] = 0;

End for

hotel = new Thread();

End Function

Public Static Void Function main(String[] args)

Hotel hotel = new Hotel();

Print "Simulation starts";

// Creating two front-desk employees.

For int i = 0 until i < 2 do i++

new FrontDesk(i, hotel);

// Creating two bellhops.

for int i = 0 until i < 2 do i++

new Bellhop(i, hotel);

// Creating 25 guests.

for int i = 0 until i < 25 do i++

new Guest(i, hotel);

// This loop ensures that program runs until

// there are guests who are not in their rooms.

while(Hotel.guestJoined < 25)

Print “”

Print Simulation ends;

Exit Program

End Function

Public Static void joinedGuests() // When guests are joined.

++Hotel.guestJoined;

End Function

Public Void Function Run()

End Function

End Class

**Guest Class**

**Class Guest implements Runnable**

Public Hotel hotel;

Public int id; // Guest id.

Public int roomNumber; // The guest's room number.

Public int bagCount; // Count of bags the guest has.

Public Thread guest;

Public Constructor Function Guest(int ID, Hotel hotel) // Initializing all of the required variables for each instance of Guest.

Random random = new Random();

this.hotel = hotel;

id = ID;

bagCount = random.nextInt(5 - 0 + 1);

Print "Guest " + id + " created");

guest = new Thread(this);

guest.start();

End Function

Public Void Function Run()

Try

Print "Guest " + id + " enters the hotel with "

+ bagCount + " bags");

// Adding guest to queue to the front desk while mutual exclusion.

wait(Hotel.guestMutex);

Hotel.guestQueue.add(this);

signal(Hotel.guestMutex);

// Wait for an available front desk employee, then receive a room.

wait(Hotel.frontDeskEmp);

signal(Hotel.guestAvailable);

wait(Hotel.guestsCompleted[id]);

wait(Hotel.roomAssigned);

Print "Guest " + id + " receives room key for room " + roomNumber

+ " from front desk employee " + Hotel.frontDeskID[id]);

// Indicate that guest is walking away from the front desk employee.

signal(Hotel.leftFrontDesk);

// Guests with more than 2 bags requests a bellhop.

if more than two bags for a guest

// Guest requests help with bagCount and waits for a bellhop.

wait(Hotel.bellhop);

Print "Guest " + id + " requests help with bags");

Hotel.bellhopQueue.add(this);

signal(Hotel.bellhopRequested);

// Guest goes to their room.

wait(Hotel.bagsReceived);

Print "Guest " + id + " enters room " + roomNumber;

signal(Hotel.reachedRoom[id]);

// Wait for the bellhop to bring bagCount before going to bed.

wait(Hotel.bagsDelivered);

Print "Guest " + id + " receives bags from bellhop "

+ Hotel.bellhopID[id] + " and gives tip");

End if

else // Guest with fewer than 2 bags goes to their room.

signal(Hotel.reachedRoom[id]);

Print "Guest " + id + " enters room " + roomNumber);

End else

Print "Guest " + id + " retires for the evening"); // Now the guest can retire.

End Try

Catch Exception e

e.printStackTrace();

End Catch

try // Join the thread.

Hotel.joinedGuests();

Print "Guest " + id + " joined");

guest.join();

End Try

catch (InterruptedException e)

e.printStackTrace();

End Catch

End Function

End **Class**

**Front Desk Class**

Class FrontDesk implements Runnable

Public int id; // Front desk employee id.

Public Static int roomNumber = 0; // Assigned rooms for the guests.

Public Function FrontDesk (int num, Hotel hotel) // Initializing all the required variables.

id = num;

Print "Front desk employee " + id + " created");

Thread frontDesk = new Thread(this);

frontDesk.start();

End Function

Public Function run()

try

While true

wait(Hotel.guestAvailable) // Wait for guest.

wait(Hotel.frontDeskMutex); // Give guest a room without interrupts.

Guest guest = Hotel.guestQueue.remove();

roomNumber++;

guest.roomNumber = roomNumber; // Store room number.

signal(Hotel.frontDeskMutex)

Hotel.frontDeskID[guest.id] = id; // Store front desk employee id for the guest.

signal(Hotel.roomAssigned)

Print "Front desk employee " + id +

" registers guest " + guest.id +

" and assigns room " + guest.roomNumber);

signal(Hotel.guestsCompleted[guest.id]) // Guest completed.

wait(Hotel.leftFrontDesk) // Guest leaves front desk.

signal(Hotel.frontDeskEmp) // One front desk employee is freed.

End While

End Try

Catch Exception e

e.printStackTrace();

End catch

End Function run

End Class FrontDesk