The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

(1) **Semaphores**

- static Semaphore *guestMutex* = new Semaphore 1 ;
    - When a guest enters the hotel, it tries to acquire guestMutex to add itself to the queue. This ensures that only one guest can access the queue at a time to avoid race conditions. After a guest enters the queueForFrontDesk, it releases the mutex for the next guest to use.
    - Signaled by Guest
    - Waited by Guest
    - This semaphore is initially set to 1 because it is a binary semaphore, if the guest mutex is “locked” it will be set to 0, if “unlocked” and available, it will be set to 1.

- static Semaphore *employeeMutex* = new Semaphore 1 ;
    - When an employee is assigning a room to the guest during the check in process, it tries to acquire employeeMutex. This mutex is used to ensure that only one employee is trying to assign a room to the same guest at the same time.
    - Signaled by Employee
    - Waited by Employee
    - This semaphore is initially set to 1 because it is a binary semaphore, if the employee mutex is “locked” it will be set to 0, if “unlocked” and available, it will be set to 1.

- static Semaphore *bellhopMutex* = new Semaphore 1 ;
    - When a bellhop is called to get bags from the guest, it tries to acquire the bellhopMutex to ensure that only one bellhop attempts to deliver bags to the same guest at the same time.
    - Signaled by Bellhop
    - Waited by Bellhop
    - This semaphore is initially set to 1 because it is a binary semaphore, if the bellhop mutex is “locked” it will be set to 0, if “unlocked” and available, it will be set to 1.

- static Semaphore *employees* = new Semaphore 2 ;
    - This semaphore initializes how many front desk employees that can operate concurrently (2). There can not be more than 2 front desk employees operating.
    - Signaled by Employee

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

  - Waited by Guests
  - This semaphore is initialized to 2 because it allows up to 2 threads to acquire it simultaneously.

- static Semaphore *bellhops* = new Semaphore(2);
  - This semaphore initializes how many bellhops that can operate concurrently (2). There can not be no more than 2 bellhops operating.
  - Signaled by Bellhop
  - Waited by Guests
  - This semaphore is initialized to 2 because it allows up to 2 threads to acquire it simultaneously.

- static Semaphore *guestsWaitingForCheckIn* = new Semaphore(0);
  - This semaphore is used to notify employees that there are guests in line waiting to be checked in. Once guests have joined the queue, they signal this semaphore. Employees are synchronized on this semaphore, allowing them to process guests by dequeuing them from the queue.
  - Signaled by Guests
  - Waited by Bellhop
  - This semaphore is initialized to 0 because at the start of the simulation, there are no guests waiting for check in. This semaphore is incremented which signals when a guest is ready.

- static Semaphore *frontDeskAvailable* = new Semaphore(0);
  - This semaphore enables employees to assist other guests once the current guest has completed the check-in process at the front desk. Employees wait on this semaphore, and upon acquiring it, they signal their availability through the `employees` semaphore, allowing them to attend to the next guest in the queue. This ensures a smooth and continuous flow of guest check-ins.
  - Signaled by Guests
  - Waited by Employees

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

    - This semaphore is initialized with 0 because the front desk is only available when an employee is available.

- static Semaphore *employeeChecksGuestIn[]* = new Semaphore 0 ;
    - This array of semaphores allows for each guest to be signaled by an employee to notify that the guest has been assigned a room. The guest waits for this semaphore to receive a room key.
    - Signaled by Employee
    - Waited by Guests
    - This semaphore is used to ensure that an employee checks in a guest before the guest proceeds further. It starts at 0 because initially, no guests have been checked in. It will be incremented when an employee successfully checks in a guest.

- static Semaphore *requestForBellhop* = new Semaphore 0 ;
    - If guests have more than 2 bags, this semaphore is signaled by the guest. The bellhop thread waits for this semaphore and will assist the guests once signaled.
    - Signaled by Guests
    - Waited by Bellhop
    - This semaphore signals when a guest requests help from a bellhop if they have more than 2 bags. It starts at 0 because initially, no guest has requested bellhop assistance. It will be incremented, which signals when a guest requests help.

- static Semaphore *bagsReceived* = new Semaphore 0 ;
    - This semaphore signals that the bellhop has received the bags from the guest. The guests wait for this signal to then enter their assigned room.
    - Signaled by Bellhop
    - Waited by Guests
    - This semaphore is initially 0 because at the beginning of the simulation, no guest has given the bellhop any bags. It signals that the bellhop has received the bags from the guest by incrementing.

- static Semaphore *bagsDelivered* = new Semaphore 0 ;

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

  - This semaphore signals that the bellhop has completed their delivery to the guest. The guest waits for this signal so they know when to tip the bellhop and then proceed with their stay. This semaphore cannot happen until the guests enter their room. After bagsDeliveredByBellhop is signaled, bellhop releases the semaphore twoBellhops to indicate that there is an available bellhop and the bellhop receives a tip from the guest.
  - Signaled by Bellhop
  - Waited by Guest
  - This semaphore is initially 0 because at the beginning of the simulation, no bags have been delivered to the guest. It signals that the bellhop has delivered the bags to the guest by incrementing.

- static Semaphore *guestEnteredRoom[]* = new Semaphore 0 ;
  - This array of semaphores ensures that the guests do not enter their room until it is ready, also helping ensure that the bellhops deliver the bags to the room after the guests have entered their room.
  - Signaled by Guests
  - Waited by Bellhop
  - This semaphore is initialized with 0 as at the beginning of the simulation, no rooms are available for guests who are not there. Once guests have enter and have been checked in, their room will be available to them.

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

(2) **Pseudocode**

```
/*program Hotel*/
public class Project2 {
        public static void main(String[]args){
            Print("Simulation starts");

            Hotel hotel = new Hotel();

            for(i to 2){
               new Employee(i);
            }

            for(i to 2){
               new Bellhop(i);
            }

            for(i to 25){
               new Guest(i);
            }

            for(i to 25){
               Hotel.hotelGuestThreads[i].join();
                        Print("Guest " + i + " joined");

            }

            if(Hotel.joinedGuests == Hotel.MAX_GUESTS) {
                Print("Simulation ends");
            }

            System.exit(0);
         }
}
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
public class Hotel implements Runnable{

   final static int MAX_GUESTS = 2;

   Semaphore guestMutex = 1;
   Semaphore employeeMutex = 1;
   Semaphore bellhopMutex = 1

   Semaphore employees = 2;
   Semaphore bellhops = 2

   Semaphore guestsWaitingForCheckIn = 0;
   Semaphore frontDeskAvailable  = 0;
   Semaphore employeeChecksGuestIn[] = 0;

   Semaphore requestForBellhop = 0;
   Semaphore bagsReceived = 0;
   Semaphore bagsDelivered = 0;
   Semaphore guestEnteredRoom[] = 0;

   int frontDeskIDs[] = int[MAX_GUESTS];
   int bellhopIDs[] = int[MAX_GUESTS];

   Initialize queueForFrontDesk as an empty ArrayDeque
   Initialize queueForBellhop as an empty ArrayDeque

   static Thread hotelGuestThreads[] = new Thread[MAX_GUESTS];
   private Thread hotelThread;

   public Hotel (){
      start Hotel thread
   }
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
    public void run (){
            initialize ID arrays
    }


}
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
class Employee implements Runnable{
  int frontDeskID;
  int roomNumber = 0;
  Thread frontDeskThread;

// parameterized constructor implementation
  public Employee(int newID){
    frontDeskID = newID;
    Print("Front desk employee " + frontDeskID + " created");
    Start front desk thread
  }

// run method implementation
  @Override public void run(){
      while(true){
            wait(guestsWaitingForCheckIn);
            wait(employeeMutex);
            Guest guest = Hotel.queueForFrontDesk.remove();
            roomNumberForFrontDesk++;
            guest.roomNumberForGuest = roomNumberForFrontDesk;
            signal(employeeMutex);
            Hotel.frontDeskIDs[guest.guestID] = frontDeskID;
            signal(employeeChecksGuestIn);
          employeeRegistersGuestAndAssignsRoom(frontDeskID,guest.guestID,guest.roomNumberForGu
          est);
            wait(frontDeskAvailable);
            signal(employees)
          }
    }
  }

  public void employeeRegistersGuestAndAssignsRoom(int employeeID, int guestID, int roomNumber) {
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
        Print("Front desk employee " + employeeID + " registers guest " + guestID +
            " and assigns room " + roomNumber);
  }
}
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
class Bellhop implements Runnable{


  int bellhopID;
  Thread bellhopThread;



  public Bellhop(int newID){
    bellhopID = newID;
    Print("Bellhop " + bellhopID + " created");
    Create and start bellhop thread
  }

// run method implementation
  @Override public void run(){

      while(true){
            signal(requestForBellhop);
            wait(bellhopMutex);
            Guest guest = Hotel.queueForBellhop.remove();
            Hotel.bellhopIDs[guest.guestID] = bellhopID;
            signal(bellhopMutex)
            bellhopReceivesBags(bellhopID, guest.guestID);
            signal(bagsReceived)
            wait(roomReadyForGuest)
            bellhopDeliversBags(bellhopID, guest.guestID);
            signal(bagsDelivered);
            signal(bellhops)
      }

  }

  public void bellhopReceivesBags(int bellhopID, int guestID) {
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
        Print("Bellhop " + bellhopID +
                " receives bags from guest " + guestID);
  }

  public void bellhopDeliversBags(int bellhopID, int guestID) {
        Print("Bellhop " + bellhopID +
                " delivers bags to guest " + guestID);
  }
}
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
class Guest implements Runnable{
// attributes
  int guestID;
  int roomNumber;
  int numBags;

  Hotel h;
  Thread guestThread;



// parameterized constructor implementation
  public Guest(int newID){
    guestID = newID;
    create random numBags
    Print("Guest " + guestID + " created");
    guestThread = new Thread(this);
    Hotel.hotelGuestThreads[guestID] = guestThread;
    start guest thread
  }

  // run method implementation
   @Override public void run(){
    try{
      guestEntersHotel(guestID, numBags);
      wait(guestMutex)
      Hotel.queueForFrontDesk.add(this);
      Signal(guestMutex.release)
      Wait(employees)
      Signal(guestsWaitingForCheckIn)

      Wait(employeeChecksGuestIn)
      guestReceivesKey(guestID, roomNumber,
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
                    Hotel.frontDeskIDs[guestID]);
    Signal(frontDeskAvailable)

    // if numBags requires help from bellhop
    if(numBags > 2){
                    wait(bellhops)
                    guestRequestsBagHelp(guestID);
                    Hotel.queueForBellhop.add(this);
                    Signal(requestForBellhop)
                    Wait(bagsReceived)
                    guestEntersRoom(guestID, roomNumber);
                    signal(roomReadyForGuest)
                    wait(bagsDelivered)
                    guestReceivesBagsAndGivesTip(guestID, Hotel.bellhopIDs[guestID]);
    }
    // if numBags is less than 2
    else{
                    signal(roomReadyForGuest)
                    guestEntersRoom(guestID, roomNumber);
    }
    guestRetires(guestID);
    guestJoins(guestID);
  }
  catch(Exception e){
    e.printStackTrace();
  }
}

 public void guestEntersHotel(int guestID, int numBags){
  Print("Guest " + guestID + " enters the hotel with " +
                    numBags + " bags");
 }
```

The design should consist of two things: (1) a list of every semaphore, its purpose, and its initial value, and (2) pseudocode for each function. The pseudocode should be similar to the pseudocode shown in the textbook for the barbershop problem. Every wait and signal call must be included in the pseudocode.

```
  public void guestReceivesKey(int guestID, int roomNumber, int employeeID){
    Print("Guest " + guestID + " receives room key for room " +
                                roomNumber + " from front desk employee " +
                                employeeID);
  }

  public void guestRequestsBagHelp(int guestID){
    Print("Guest " + guestID + " requests help with bags");
  }

  public void guestEntersRoom(int guestID, int roomNumber){
    Print("Guest " + guestID + " enters room " + roomNumber);
  }

  public void guestReceivesBagsAndGivesTip(int guestID, int bellhopID){
    Print("Guest " + guestID + " receives bags from bellhop " +
                                bellhopID + " and gives tip");
  }


  public void guestRetires(int guestID){
    Print("Guest " + guestID + " retires for the evening");
  }


  public void guestJoins(int guestID){
    Hotel.joinedGuests = Hotel.joinedGuests + 1;
  }

}
```