```
public class Customer {
   private int load;

   public Customer() {
      load = Poisson.poisson();
   }

   public String toString() {
      return "" + load;
   }

   public void unload() {
      load--;
   }

   public boolean done() {
                                                // Uppgift  A8
   }

   public static void main(String[] args) {
     Customer c = new Customer();
     while (!c.done()){
        c.unload();
        System.out.print(c + " ");
     }
     System.out.println();
   }
}

/* Output:
 4 3 2 1 0
 */



/////////////////////////////////////////////////

public class Desk {
  private ArrayList<Customer> queue;
  private boolean open;

  public Desk() {
                                                // Uppgift A9
  }

  public String toString() {
     if (open) {
        return "Open  " + queue.toString();
     } else {
        return "Closed ";
     }
  }

  public void open() {
                open = true;
  }

  public void open(ArrayList<Customer> q) {
    open = true;
    queue = q;
  }

  public boolean isOpen() {
    return open;
  }

  public void add(Customer c) {
                                                // Uppgift A10
  }
```

```

  public int queueLength () {
    return queue.size();
  }

  public void step() {
                                                // Uppgift B1
  }

  public ArrayList<Customer> removeHalfQueue() {
                                                // Uppgift B2
  }

  public static void main(String[] args) {
    Desk d = new Desk();
    System.out.println(d);
    d.open();
    d.add(new Customer());
    d.add(new Customer());
    d.add(new Customer());
    while (d.queueLength() > 0) {
       d.step();
       System.out.println(d);
    }
  }
}

/* Output:
Closed
Open   [1, 3, 1]
Open   [0, 3, 1]
Open   [3, 1]
Open   [2, 1]
Open   [1, 1]
Open   [0, 1]
Open   [1]
Open   [0]
Open   []
 */



///////////////////////////////////////////////

public class Store {
   private Desk[] theDesks;

   /**
    * Create a store with n desks.
    * The first desk (index 0) should be open, the other closed
    */
   public Store(int n) {
                                                // Uppgift A11
   }

   public void print() {
      for (int d = 0; d < theDesks.length; d++) {
         System.out.println(d + " " + theDesks[d]);
      }
   }

   /**
    * Find and return the length of the longest queue
    */
   public int maxQueue() {
      int result = 0;
      for(Desk d: theDesks) {
         if (d.isOpen()) {
            result = Math.max(result, d.queueLength());
         }
      }
      return result;
   }

   /* Find the first (i.e. the one with thw lowest
    * number) closed desk and return it’s index.
    * Return −1 if no closed desk is found. */
   private int findFirstClosed() {
                                                // Uppgift A12
   }

   /**
    * Open the first closed desk.
    * Do nothing if there are no closed desks.
    * If the desk found has index greater than 0,
    * move the second half of the queue from the
    * desk immediately befor to this one.
    */
   public void openNewDesk() {
                                                // Uppgift B3
   }

   /**
    * Returns the open desk with the shortest queue
    */
   public Desk findShortest() {
                                                // Uppgift B4
   }

   /**
    * Make all desks take one time step
    */
   public void step() {
      for (Desk d: theDesks) {
         d.step();
      }
   }

   public static void main(String[] args) {
      Store store = new Store(3);

      for (int time = 1; time<=10; time++) {
         if (store.maxQueue() > 3) {
            store.openNewDesk();
         }
         if (Math.random()<0.5) {
            store.findShortest().add(new Customer());
         }
         store.step();
         System.out.println("Time: " + time);
         store.print();
         System.out.println();
      }
   }
}
/* Output
Time: 1
0 Open   []
1 Closed
2 Closed

Time: 2
0 Open   []
1 Closed
2 Closed

Time: 3
0 Open   []
```

```
1 Closed
2 Closed

Time: 4
0 Open   [2]
1 Closed
2 Closed

Time: 5
0 Open   [1, 5]
1 Closed
2 Closed

Time: 6
0 Open   [0, 5, 8]
1 Closed
2 Closed

Time: 7
0 Open   [5, 8, 6]
1 Closed
2 Closed

Time: 8
0 Open   [4, 8, 6, 4]
1 Closed
2 Closed

Time: 9
0 Open   [3, 8]
1 Open   [5, 4]
2 Closed

Time: 10
0 Open   [2, 8]
1 Open   [4, 4, 6]
2 Closed
 */
```