**Question**:- How would you design a garbage collection system? The idea here is to design a system that recycles unused memory in the program. Here the key is to find which piece of memory is unused. What data structure would be chosen and how would it be used to ensure garbage collection does not halt the main process?

**Analysis:-**

**Non-Functional-Requirement**:-

2) While processing the objects system should not break the application.

3) Downtime for application should be as slow as possible.

**Functional Requirements:-**

1. There should be automated process to trigger GC (There must be some flag which raises a bar that Heap is now full make It free).
2. System should find out which all objects are not reachable and make them free from memory.
3. It should be reliable and make sure that right objects ( objects which are not reachable) should be removed.

**Technical Analysis:-**

1. We can use Queue or Stack for storing references help in access using BFS algo.
2. We can use Directed Graph Data structure to have references.
3. We can use find unreachable Objects by traversing the Graph ( BFS ) and those are left that can be removed.

**Algo Tri Colour:-**1) Mark first Node—Grey and then store its next references to Queue and then mark it black.

2) Then takes every element from queue and when all references of that element are stored in queue then mark that node as Black

3) In this way traverse the whole graph and all nodes which are white after that can be removed.

**Question**:- A barbershop consists of a waiting room with n chairs, and a barber chair for giving haircuts. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the interaction between the barber and the customers.

**Requirements:-**

1. Barber Shop🡪 chairs, barber chair
2. No Customer🡪Barer ( Sleep)
3. BarberShop🡪Chairs(Occupied) 🡪 Customer Leave
4. Barber(Busy)🡪Chairs(Available)🡪 customer sit on one free chair.
5. Barber(asleep)🡪 customer wake up

Entities:-

BarberShop-🡪 Chairs of Barber, Customer Chairs, no of Barbers

Assumption:-

No of Chairs of Barber is == No of Barber

**Analysis:-**

1. **Solution can be made by creating a fixed no of thred pools of barbers.**
2. **When seats are full at that time it should call any method.**
3. **When seats are there then any barber thread which is free can call that method.**
4. **If no customer are there then thread pools can be set to sleep state**
5. **If any customer comes then we can use notifyAll option to call the sleep thread**

**Issue is:-**

1. **Concurrency:- at a same time if two are entered in barber shop then they should see the correct data. We can achieve this by making it thread safe but our system latency goes high in that case.**

**Another way:-**

1. **Maintaining a copy of DB in in memory cache like Redis (this will make system faster response time will be slower) and in the end if any updates goes in cache same will be updated in DB using procedure.**

**Classes:-**

Class BarberShop{

List<Barber> barber;

List<Chair> chair;

// Getter and setter below

}

Public interface BarberShopFeatures {

Public Boolean wantHairCut(int customer id);

}

Public class BarberShopFeatuersImpl implements BarberShopFeatures{

Public Boolean wantHairCut(int customer id){

List<Seats> seat = getSeatStatusFromDB);

List<Seats> remaingSeats = seats.filter(x->x.getSeatStatus!=true).collect(Collections.asList());

List<Barber> freeBarber = barber.filter(x->x.getBarberStatus!=true).collect(Collections.asList());

If(remainigSeats.size() == capacityofSeats){

List<freeBarber> freeBarber = Barber.filter(x->x.setBarberStatus = True).collect(Collections.asList());

pushinDB(freeBarber);

}

If(reamingSeats.size()>0 && freeBarber==0 ){

remaingSeats.getElement(0).setSeatStatus(True).

PushinDB(list<seats);

}

Else if(freeBarber>0){

Barber.get(0).setisFreeBarber(true);

PushinDB(Barber);

}

Else{

Return false;

}

Return true;

}

}

**Question:-** Design an API Rate limiting system that monitors the number of requests per a window time

System should be designed considering the following:  
  
  a)  Rate limiting should work for a distributed set up as the APIs are available through a group of API Gateways

 b)  What database would be used and the rationale behind the choice

  c)  How would throttling be done?

  d) The system should be highly available

1. For this I think token based approach will work

**Analysis:-**

1. **We should have in memory bucket to store the request and the count of no of request.**
2. **When the first request comes it store the token in particular format like:- timestamp+date along with updated token and count.**

**Design should be:-**

**User id First token subsequent token count .**

1. **If request lies with in the time limit of first token and count will be less the given count then we will increase the count and mark the subsequent token the latest time stamp.**
2. **If request comes outside the bracket of duration then we will create an another entry of first token and then allocate the count++.**
3. **If request comes with in the limit but count has reached upto limit then we can raise exception and send it back to server.**
4. **For in memory data storage we can use Redis as key value pair.**