ASSIGNMENT # 1: OBJECT ORIENTED PROGRAMMING

**HOSPITAL MANAGEMENT SYSTEM**

**SUBMITTED TO:**

DR. XYZ

**DATE:**

02-April-19

**SECTION:**

XYZ

**SUBMITTED BY:**

XYZ - ID

**CASE STUDY - HOSPITAL MANAGEMENT SYSTEM**

A new hospital is seeing an increase in the number of their patients. They want to create a software system that manage their employees and patients. A part of their software requirement is given below. You also are required to do a self-study on how the out-patient departments of hospitals function and create a design for managing the department.

# **PROBLEM ANALYSIS:**

**REQUIREMENTS:**

1. Hospital containing people that can either be patients or staff members
2. Staff members work in shifts of 8 hours per day with every staff member working in one shift only.
3. Each day, the first shift begins at 7am
4. Staff members can be either administrative, operations or technical.
5. Every staff member has to mark his attendance when he arrives.
6. Administrative Staff Member is responsible for registering patients and maintaining record of the patients in the waitingQueue anf finally generating bill for them to pay.
7. Operational Staff Member is responsible for getting the patient checked and diagnosing his disease
8. Technical Staff Member is responsible for finding and fixing defects in the machines on site
9. At any point in time, the system should be able to display details of staff members of the ongoing shift and patients in the hospital.

**LIST OF CLASSES:**

* Hospital
* Person
* Patient
* StaffMember
* Shift
* AdministrativeStaffMember
* OperationalStaffMember
* TechnicalStaffMember
* Machine

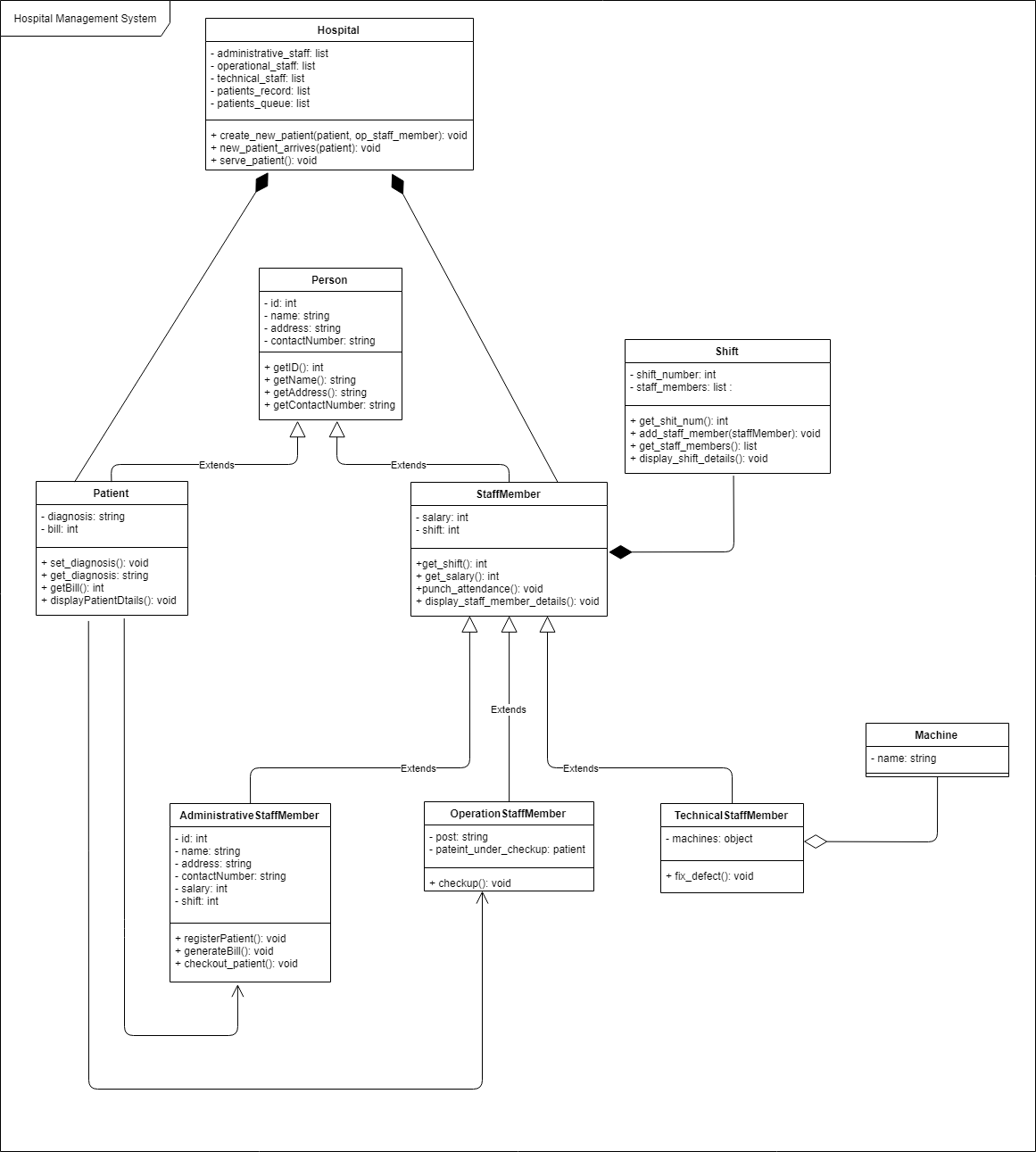
# **FUNCTIONAL DESIGN:**

**LOGICAL FLOW:**

StaffMember arrives at his shift, marks attendance, performs his specified tasks until his shift ends. Patient arrives at the reception desk of the administrative Staff Member, registers himself, waits in the waitingQueue, goes for checkup to the operational Staff Member on his turn, gets checked up, returns to administrative Staff Member who generates his bill, patient pays dues and exits the hospital. While the technical staff Member performs his task of defect fixing side by side.

# **class design:**

**UML DIAGRAM:**



# **SOURCE CODE:**

import datetime, time, random

class Hospital:

def \_\_init\_\_(self, administrative\_staff: list, operational\_staff: list, technical\_staff: list):

self.administrative\_staff = administrative\_staff

self.operational\_staff = operational\_staff

self.technical\_staff = technical\_staff

self.patients\_record = {}

self.patients\_queue = []

def create\_new\_patient(self, patient, op\_staff\_member):

self.patients\_record[patient] = op\_staff\_member

def new\_patient\_arrives(self, patient):

admin\_staff\_member = random.choice(self.administrative\_staff)

op\_staff\_member = random.choice(self.operational\_staff)

admin\_staff\_member.register\_patient(self, patient, op\_staff\_member)

self.patients\_queue.append(patient)

return

def serve\_patient(self):

patient = self.patients\_queue.pop(0)

staff\_assigned = self.patients\_record[patient]

staff\_assigned.checkup(patient)

self.patient\_leaves(patient)

def patient\_leaves(self, patient):

admin\_staff\_member = random.choice(self.administrative\_staff)

admin\_staff\_member.checkout\_patient(patient)

class Shift:

shift\_number = None

staff\_members = []

def \_\_init\_\_(self, shift\_number):

if shift\_number < 0:

shift\_number = -shift\_number

self.shift\_number = (shift\_number + 3) % 3

# self.staffPerShift = staffPerShift

def get\_shift\_num(self):

if self.shift\_number == 1:

print("Shift 1. 7AM to 3PM")

elif self.shift\_number == 2:

print("Shift 2. 3PM to 11PM")

elif self.shift\_number == 3:

print("Shift 3. 11PM to 7AM")

return self.shift\_number

def add\_staff\_member(self, staff\_member):

self.staff\_members.append(staff\_member)

def get\_staff\_members(self):

return self.staff\_members

# Display Shift

def display\_shift\_details(self):

print("Shift {}".format(self.shift\_number))

for v in self.staff\_members:

v.displayStaffMemberDetails()

class Person:

def \_\_init\_\_(self, person\_id, name, address, contact\_number):

self.person\_id = person\_id

self.name = name

self.address = address

self.contact\_number = contact\_number

def get\_id(self):

return self.person\_id

def get\_name(self):

return self.name

def get\_address(self):

return self.address

def get\_contact\_number(self):

return self.contact\_number

class Patient(Person):

def \_\_init\_\_(self, patient\_id, name, address, contact\_number):

super(Patient, self).\_\_init\_\_(patient\_id, name, address, contact\_number)

self.diagnosis = ""

self.bill = 0

def set\_daignosis(self, diagnosis):

self.diagnosis = diagnosis

def set\_bill(self, bill):

self.bill = bill

def get\_diagnosis(self):

return self.diagnosis

def get\_bill(self):

return self.bill

def display\_patient\_details(self):

patient\_details = ""

patient\_details += str(self.get\_id()) + '\t'

patient\_details += self.get\_name() + '\t'

patient\_details += self.get\_address() + '\t'

patient\_details += str(self.get\_contact\_number()) + '\t'

patient\_details += self.get\_diagnosis() + '\t'

patient\_details += str(self.get\_bill())

return patient\_details

# print(patient\_details)

class StaffMember(Person):

def \_\_init\_\_(self, staff\_id, name, address, contact\_number, salary, shift: Shift):

super(StaffMember, self).\_\_init\_\_(staff\_id, name, address, contact\_number)

self.salary = salary

self.shift = shift

def get\_salary(self):

return self.salary

def get\_shift(self):

return self.shift

def punch\_attendance(self):

if self in self.shift.get\_staff\_members():

return

self.shift.add\_staff\_member(self)

print("Attendance marked. Added to shift {}.".format(self.get\_shift().get\_shift\_num()))

def display\_staff\_member\_details(self):

details = ""

details += str(self.get\_id()) + '\t'

details += self.get\_name() + '\t'

details += self.get\_address() + '\t'

details += str(self.get\_contact\_number()) + '\t'

details += str(self.get\_salary()) + '\t'

details += str(self.shift.get\_shift\_num()) + '\t'

print(details)

class AdministrativeStaffMember(StaffMember):

def \_\_init\_\_(self, staff\_id, name, address, contact\_number, salary, shift):

super(AdministrativeStaffMember, self).\_\_init\_\_(staff\_id, name, address, contact\_number, salary,

shift)

def register\_patient(self, hospital: Hospital, patient, operational\_staff\_member):

hospital.create\_new\_patient(patient, operational\_staff\_member)

print("Patient: {} registered by {}.".format(patient.get\_name(), self.get\_name()))

return

def generate\_bill(self, patient):

disease1 = "Cancer"

disease2 = "Diabetes"

disease3 = "Fever"

if disease1 in patient.diagnosis:

patient.set\_bill(20000)

elif disease2 in patient.diagnosis:

patient.set\_bill(4000)

elif disease3 in patient.diagnosis:

patient.set\_bill(1000)

print("Bill generated for patient {}. Generated by: {}".format(patient.get\_name(), self.get\_name()))

def checkout\_patient(self, patient):

self.generate\_bill(patient)

class OpeationalStaffMemeber(StaffMember):

def \_\_init\_\_(self, staff\_id, name, address, contact\_number, salary, shift, post):

super(OpeationalStaffMemeber, self).\_\_init\_\_(staff\_id, name, address, contact\_number, salary, shift)

self.post = post

self.patient\_under\_checkup = None

def checkup(self, patient):

print("Treatement started b {} {}. Patient details: {} \n".format(self.post,

self.get\_name(),

patient.display\_patient\_details()))

diseases = ["Cancer", "Diabetes", "Fever"]

disease = random.choice(diseases)

patient.set\_daignosis(disease)

print("Patient: {}. Disease diagnosed: {}".format(patient.get\_name(), disease))

class Machine:

def \_\_init\_\_(self, name):

self.name = name

class TechnicalStaffMember(StaffMember):

def \_\_init\_\_(self, staff\_id, name, address, contact\_number, salary, shift, machines):

super(TechnicalStaffMember, self).\_\_init\_\_(staff\_id, name, address, contact\_number, salary, shift)

self.machines = machines

def fix\_defect(self):

for machine in self.machines:

print("Fixing defect of {} machine.".format(machine.name))

time.sleep(1)

# **CONCLUSION:**

The case study was helpful to learn relationships between classes and strengthened understanding of core concepts of object oriented programming i.e. inheritance, encapsulation, aggregation, composition and association.

**PROBLEMS FACED:**

Finding out correct relationships between classes and their interlinking between one another.

**FUTURE EXPANSION:**

Designing a real-time hospital management application linked to a database.