Assignment No 01

Doctor 3 Tier

DoctorModelLibrary

## Doctor.cs

## public int Id { get; set; }

public string Name { get; set; }

public int YearOfExperience { get; set; }

public string Specialization { get; set; }

public void TakeDoctorDeatilsFromConsole()

{

Console.WriteLine("Please enter the doctor's name");

Name = Console.ReadLine();

Console.WriteLine("Please enter the Year Of Experience");

YearOfExperience = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter The Specialization doctor's have");

Specialization = Console.ReadLine();

}

public override string ToString()

{

return $"Doctor Id : {Id} \n " +

$"Doctor Name : {Name} \n" +

$"Doctor Year Of Experience : {YearOfExperience} \n" +

$"Doctor Type : {Specialization}";

}

public override bool Equals(object? obj)

{

Doctor d1, d2;

d1 = (Doctor)obj;

d2 = this;

if (d1.Id == d2.Id)

return true;

return false;

}

}

}

## DoctorsApplications

### Program.cs

using DoctorBULLibrary;

using DoctorDALLibrary;

using DoctorsModelLibrary;

namespace DoctorsApplications

{

internal class Program

{

ManageDoctor manageDoctor = new ManageDoctor();

void InitializingApplication()

{

int choice = 0;

do

{

Console.WriteLine("Welcome to the Doctors Data Application");

Console.WriteLine("1. Enter the Doctors name");

Console.WriteLine("2. List the Doctors");

Console.WriteLine("3. List Doctor by Specialization");

Console.WriteLine("4. List Doctor by YearOFExperience");

Console.WriteLine("0. Exit App");

Console.WriteLine("Enter your choice");

choice = Convert.ToInt32(Console.ReadLine());

switch (choice)

{

case 0:

Console.WriteLine("Byeeeeeee");

break;

case 1:

InitiazeDoctorsName();

break;

case 2:

ListOfDoctors();

break;

case 3:

ListBySpecialization();

break;

case 4:

ListByDoctorExperience();

break;

default:

Console.WriteLine("Invalid choice");

break;

}

} while (choice != 0);

}

private void ListBySpecialization()

{

Console.WriteLine("Please enter the Specialization of Doctor ");

try

{

string type = Console.ReadLine().ToLower();

List<Doctor> doctors = manageDoctor.GetDoctorByType(type).ToList();

PrintDoctors(doctors);

}

catch (DoctorsNotAvailableException dnae)

{

Console.WriteLine(dnae.Message);

}

catch (InvalidDoctorException ide)

{

Console.WriteLine(ide.Message);

}

}

private void ListByDoctorExperience()

{

try

{

Console.WriteLine("Please enter the min YearOfExperience");

int min = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Please enter the max YearOfExperience");

int max = Convert.ToInt32(Console.ReadLine());

List<Doctor> doctors = manageDoctor.GetDoctorsByYearOfExperience(min, max).ToList();

PrintDoctors(doctors);

}

catch (DoctorsNotAvailableException dnae)

{

Console.WriteLine(dnae.Message);

}

catch (InvalidDoctorException ide)

{

Console.WriteLine(ide.Message);

}

}

void PrintDoctors(ICollection<Doctor> doctors)

{

foreach (Doctor p in doctors)

{

Console.WriteLine(p);

}

}

void ListOfDoctors()

{

List<Doctor> doctors = manageDoctor.GetDoctor().ToList();

PrintDoctors(doctors);

}

void InitiazeDoctorsName()

{

do

{

Console.WriteLine("Please enter the Doctors Details");

Doctor doctor = new Doctor();

doctor.TakeDoctorDeatilsFromConsole();

manageDoctor.AddANewDoctor(doctor);

Console.WriteLine("Search For Anather Doctor");

string choice = Console.ReadLine().ToLower();

if (choice != "yes")

break;

} while (true);

}

static void Main(string[] args)

{

Program program = new Program();

program.InitializingApplication();

Console.ReadKey();

}

}

}

## DoctorDALLibrary

### dRepository.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DoctorDALLibrary

{

public interface dRepository<K, T>

{

List<T> GetAll();

T GetById(K key);

T Add(T item);

T Update(T item);

T Delete(K key);

}

}

DoctorsNotAvailableException.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DoctorDALLibrary

{

public class DoctorsNotAvailableException : Exception

{

string message;

public DoctorsNotAvailableException()

{

message = "No Doctors are available at this moment. Sorry";

}

public override string Message => message;

}

}

DoctorRepository.cs

using DoctorsModelLibrary;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DoctorDALLibrary

{

public class DoctorRepository : dRepository<int, Doctor>

{

List<Doctor> doctors

= new List<Doctor>();

public Doctor Add(Doctor item)

{

if (item == null)

throw new ArgumentNullException("No doctor available");

item.Id = GeterateIndex();

doctors.Add(item);

return item;

}

private int GeterateIndex()

{

int id = doctors.Count;

return ++id;

}

public Doctor Delete(int key)

{

Doctor doctor = GetById(key);

doctors.Remove(doctor);

return doctor;

}

public List<Doctor> GetAll()

{

if (doctors.Count == 0)

throw new DoctorsNotAvailableException();

return doctors;

}

public Doctor GetById(int key)

{

Doctor doctor = doctors.FirstOrDefault(p => p.Id == key);

if (doctor == null)

throw new InvalidOperationException("No doctor with id " + key);

return doctor;

}

public Doctor Update(Doctor item)

{

Doctor doctor = GetById(item.Id);

doctor.Name = item.Name;

doctor.YearOfExperience = item.YearOfExperience;

doctor.Specialization = item.Specialization;

return doctor;

}

}

}

DoctorBULLibrary.cs

ManageDoctor.cs

using DoctorDALLibrary;

using DoctorsModelLibrary;

namespace DoctorBULLibrary

{

public class ManageDoctor

{

dRepository<int, Doctor> repository = new DoctorRepository();

public ICollection<Doctor> GetDoctor()

{

return repository.GetAll();

}

public ICollection<Doctor> GetDoctorByType(string type)

{

var doctors = GetDoctor();

if (type == "General"|| type =="pediatrition"|| type == "Surgen" || type== "Cardiologist" || type == "Psychitrist"||type =="orthopadic"||type=="Dentist" || type == "Neurologist" || type == "Urologist" || type == "Dematologist")

{

var typeSpecificDoctor =doctors.Where(p => p.Specialization == type).ToList();

if (typeSpecificDoctor.Count == 0)

throw new DoctorsNotAvailableException();

return typeSpecificDoctor;

}

throw new InvalidDoctorException();

}

public ICollection<Doctor> GetDoctorsByYearOfExperience(int min, int max)

{

var doctors = GetDoctor();

if (min >= 00 || max > 0)

{

var experienceInRange = doctors.Where(p => p.YearOfExperience >= min && p.YearOfExperience <= max).ToList();

if (experienceInRange.Count == 0)

throw new DoctorsNotAvailableException();

return experienceInRange;

}

throw new InvalidDoctorException();

}

public Doctor AddANewDoctor(Doctor doctor)

{

repository.Add(doctor);

return doctor;

}

public object GetDoctorBySpecialization(string? type)

{

throw new NotImplementedException();

}

}

}

InvalidDoctorExceptio.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DoctorBULLibrary

{

public class InvalidDoctorException: Exception

{

string message;

public InvalidDoctorException()

{

message = "No such type of Doctor ";

}

public override string Message => message;

}

}