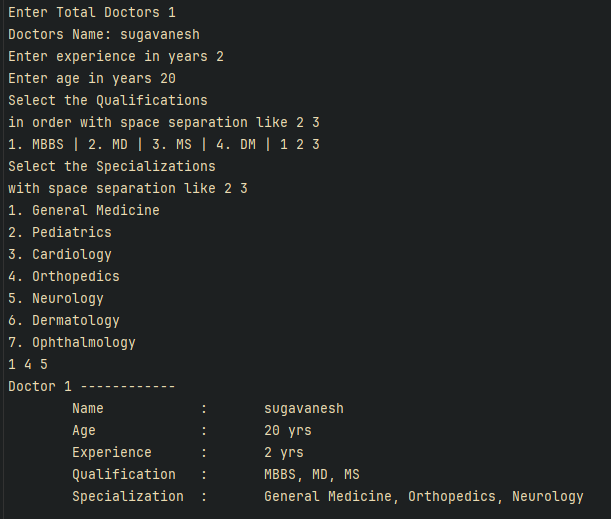
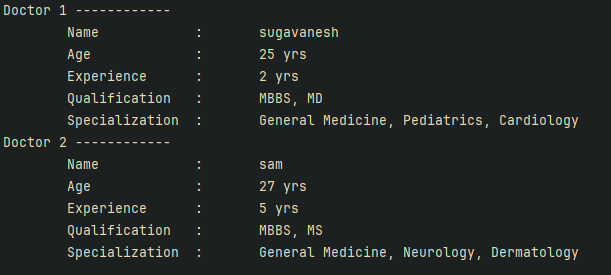
Day 4

Doctor Manager

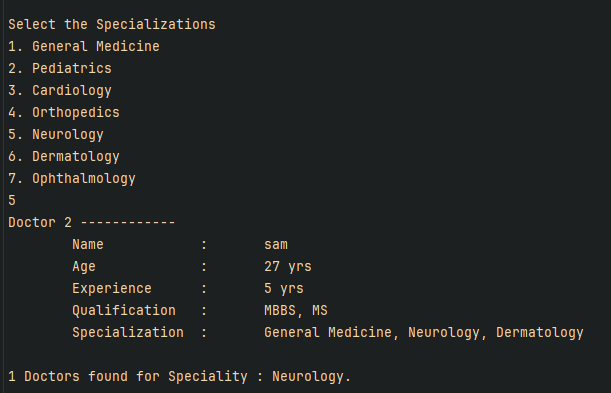
1. Creating single Doctor and displaying it

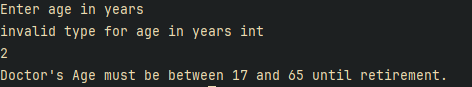
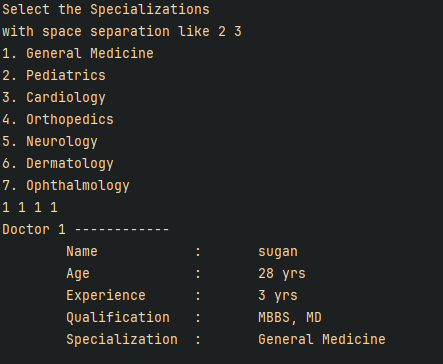


1. Creating Mutiple Doctors and displaying it



1. Searching based on Speciality



1. Validations
   1. Total No of Doctors:
      1. 
   2. For Experience:
      1. 
      2. 
   3. For Age:
      1. 
   4. Mutiple duplicate entries:
      1. 

Code:

Main Program:

class Program  
{  
 static void Main(string[] args)  
 {  
 var size = DoctorHelper.GetNum("Total Doctors");  
 var doctors = new Doctor[size];  
 for (var i = 0; i < size; i++)  
 doctors[i] = DoctorHelper.GetDoctor(i + 1);  
  
 foreach (var doctor in doctors)  
 doctor.Display();  
   
 DoctorHelper.FilterBySpeciality(doctors);  
 }  
   
}

Doctor entity:

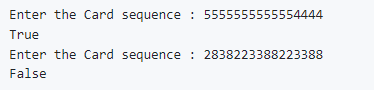
public class Doctor  
{  
  
 public Doctor(int id)  
 {  
 Id = id;  
 }  
  
 */// <summary>*  
 */// Populate doctor entity with the given fields*  
 */// </summary>*  
 */// <param name="id">Id for the doctor</param>*  
 */// <param name="name">Doctor's name</param>*  
 */// <param name="age">Age as in Int</param>*  
 */// <param name="experience">Experience as in Int</param>*  
 */// <param name="qualification">Qatlification stream</param>*  
 */// <param name="specialization">Specialization</param>*  
 */// <exception cref="ArgumentNullException"></exception>*  
public Doctor(int id, string name, int age, int experience, List<string> qualification, List<string> specialization) : this(id)  
 {  
 Name = name ?? throw new ArgumentNullException(nameof(name));  
 Age = age;  
 Experience = experience;  
 Qualification = qualification ?? throw new ArgumentNullException(nameof(qualification));  
 Specialization = specialization ?? throw new ArgumentNullException(nameof(specialization));  
 }  
  
 private int Id { get; set; }  
 public string Name { get; set; }  
   
 private int \_age;  
 public int Age   
 {   
 get => \_age;  
 set   
 {  
 if (value < 17 || value > 65) *// allowed age limits*  
{  
 throw new ArgumentException("Doctor's Age must be between 17 and 65 until retirement.");  
 }  
 \_age = value;  
 }  
 }  
  
 private int \_experience;  
 public int Experience  
 {  
 get => \_experience;  
 set  
 {  
 if (value < 0 || value > 48)  
 {  
 throw new ArgumentException("Experience cannot be negative nor greater than 48 as retirement occur.");  
 }  
 \_experience = value;  
 }  
 }   
 public List<string> Qualification { get;} = new();  
 public List<string> Specialization { get; } = new();  
   
 */// <summary>*  
 */// Method to add specialization*  
 */// </summary>*  
 */// <param name="specialization"></param>*  
public void AddSpecialization(string specialization)  
 {  
 if (Specializations.Contains(specialization))  
 {  
 if (!Specialization.Contains(specialization))  
 Specialization.Add(specialization);  
 }  
 else  
 throw new Exception($"{specialization} is not valid.");  
 }  
   
 */// <summary>*  
 */// Method to add qualification*  
 */// </summary>*  
 */// <param name="qualification"></param>*  
public void AddQualification(string qualification)  
 {  
 if (Qualifications.Contains(qualification))  
 {  
 if (!Qualification.Contains(qualification))  
 Qualification.Add(qualification);  
 }  
 else  
 throw new Exception($"{qualification} is not a valid.");  
 }  
   
 public static readonly string[] Specializations = {  
 "General Medicine",  
 "Pediatrics",  
 "Cardiology",  
 "Orthopedics",  
 "Neurology",  
 "Dermatology",  
 "Ophthalmology",  
 };  
  
 public static readonly string[] Qualifications = {  
 "MBBS",  
 "MD",  
 "MS",  
 "DM",  
 };  
  
 public void Display()  
 {  
 Console.WriteLine($"Doctor {Id} ------------");  
 Console.WriteLine($"\tName\t\t:\t{Name}");  
 Console.WriteLine($"\tAge\t\t:\t{Age} yrs");  
 Console.WriteLine($"\tExperience\t:\t{Experience} yrs");  
 var qualifications = string.Join(", ", Qualification);  
 Console.WriteLine($"\tQualification\t:\t{qualifications}");  
 var specializations = string.Join(", ", Specialization);  
 Console.WriteLine($"\tSpecialization\t:\t{specializations}");  
 }  
}

Doctor Helper:

public class DoctorHelper  
{  
 */// <summary>*  
 */// Returns Doctor objects, where info gather from user*  
 */// </summary>*  
 */// <param name="id">Id should be generated and provided</param>*  
 */// <returns>fully seeded doctor object</returns>*  
 */// <exception cref="Exception">If the name is null</exception>*  
public static Doctor GetDoctor(int id)  
 {  
 var doctor = new Doctor(id);  
 Console.Write("Doctors Name: ");  
 doctor.Name = Console.ReadLine() ?? throw new Exception("Name should not be null") ;  
   
 var validInput = false;  
  
 do  
 {  
 try  
 {  
 int experience = GetNum("experience in years");  
 doctor.Experience = experience;  
  
 int age = GetNum("age in years");  
 doctor.Age = age;  
  
 validInput = true;  
 }  
 catch (ArgumentException e)  
 {  
 Console.WriteLine(e.Message);  
 }  
 } while (!validInput);  
   
 UpdateQualification(doctor);  
 UpdateSpecialization(doctor);  
  
 return doctor;  
 }  
  
 */// <summary>*  
 */// Method to filter doctors based on given speciality*  
 */// </summary>*  
 */// <param name="doctors">Array of created doctors</param>*  
public static void FilterBySpeciality(Doctor[] doctors)  
 {  
   
 Console.WriteLine("\n\nSelect the Specializations");  
 for (var i = 0; i < Doctor.Specializations.Length; i++ )  
 Console.WriteLine($"{i + 1}. {Doctor.Specializations[i]}");  
  
 var option = Console.ReadLine();  
 if (!int.TryParse(option, out var optionPos) || optionPos > Doctor.Specializations.Length)  
 {  
 Console.WriteLine($"invalid entry {option}");  
 return;  
 }  
  
 var count = 0;  
 var specialization = Doctor.Specializations[optionPos - 1];  
 foreach (var doctor in doctors)  
 {  
 if (doctor.Specialization.Contains(specialization))  
 {  
 doctor.Display();  
 count++;  
 }  
   
 }  
  
 Console.WriteLine($"\n{count} Doctors found for Speciality : {specialization}.");  
 }  
   
 */// <summary>*  
 */// For getting qualification from cli*  
 */// </summary>*  
 */// <param name="doctor"></param>*  
  
static void UpdateQualification(Doctor doctor)  
 {  
 Console.WriteLine("Select the Qualifications\nin order with space separation like 2 3");  
 for (var i = 0; i < Doctor.Qualifications.Length; i++ )  
 {  
 Console.Write($"{i + 1}. {Doctor.Qualifications[i]} | ");  
 }  
  
 foreach (var option in Console.ReadLine()?.Split(" ")!)  
 {  
 if (!int.TryParse(option, out var optionPos) || optionPos > Doctor.Qualifications.Length )  
 {  
 Console.WriteLine($"invalid entry {option}");  
 continue;  
 }  
   
 doctor.AddQualification(Doctor.Qualifications[optionPos - 1]);  
 }  
 }  
  
 */// <summary>*  
 */// For getting Specialization from cli*  
 */// </summary>*  
 */// <param name="doctor"></param>*  
static void UpdateSpecialization(Doctor doctor)  
 {  
 Console.WriteLine("Select the Specializations\nwith space separation like 2 3");  
 for (var i = 0; i < Doctor.Specializations.Length; i++ )  
 {  
 Console.WriteLine($"{i + 1}. {Doctor.Specializations[i]}");  
 }  
  
 foreach (var option in Console.ReadLine()?.Split(" ")!)  
 {  
 if (!int.TryParse(option, out var optionPos) || optionPos > Doctor.Specializations.Length )  
 {  
 Console.WriteLine($"invalid entry {option}");  
 continue;  
 }  
 doctor.AddSpecialization(Doctor.Specializations[optionPos - 1]);  
 }  
   
 }  
  
 */// <summary>*  
 */// For getting Int type from users, until the user enter the valid one*  
 */// </summary>*  
 */// <param name="msg">Custom msg to be displayed</param>*  
 */// <returns></returns>*  
public static int GetNum(string msg)  
 {  
 int res;  
 Console.Write($"Enter {msg} ");  
 while(!int.TryParse(Console.ReadLine(), out res))  
 Console.WriteLine($"invalid type for {msg} int");  
 return res;  
 }  
   
}

Task 2: Card number validation

Output:



Code:

class Program  
{  
 static void Main(string[] args)  
 {  
 Console.Write("Enter the Card sequence : ");  
 var charSequenceString = Console.ReadLine()??"4477468343113002";  
 Console.WriteLine(ValidateCreditCardNumber(charSequenceString));  
 }  
  
 private static bool ValidateCreditCardNumber(string number)  
 {  
 var sum = 0;  
 var alternate = false;  
 int n = 16;  
  
 if (number.Length != n)  
 {  
 Console.WriteLine("card number should be equal to 16");   
 return false;  
 }  
  
 for (var i = n - 1; i >= 0; i--)  
 {  
 var digit = number[i] - '0';  
  
 if (alternate)  
 {  
 digit \*= 2;  
 if (digit > 9)  
 digit = (digit % 10) + 1;  
 }  
  
 sum += digit;  
 alternate = !alternate;  
 }  
  
 return sum % 10 == 0;  
 }  
  
}