**Name : Hasan Mahmu Roni**

**Cellphone : 01705243985**

**Email :** [**krony063@gmail.com**](mailto:krony063@gmail.com)

**Ans to the Question—1**

A.

Step 1: Identify Entities

1.Doctors

2. Service Points

3.Departments

Step 2: Define Attributes for Each Entity

\*Doctor (DoctorID, DoctorName, ContactNumber)

\* ServicePoint (ServicePointID, ServicePointName)

\* Department (DepartmentID, DepartmentName)

\*Doctor\_ServicePoint (DoctorID, ServicePointID)

\* Doctor\_Department (DoctorID, DepartmentID)

Step 3: Normalize the Data

Entities:

1. Doctor

\*DoctorID (Primary Key)

\*DoctorName

\*ContactNumber

2. ServicePoint

\*ServicePointID (Primary Key)

\*ServicePointName

3. Department

\*DepartmentID (Primary Key)

\*DepartmentName

4. Doctor\_ServicePoint (Many-to-Many Relationship)

\* DoctorID (Foreign Key referencing Doctor table)

\*ServicePointID (Foreign Key referencing ServicePoint table)

5. Doctor\_Department (Many-to-Many Relationship)

\*DoctorID (Foreign Key referencing Doctor table)

\*DepartmentID (Foreign Key referencing Department table)

This normalization divides the data into separate tables, reducing redundancy and maintaining relationships between entities.

B.

.Entity Relationship Diagram (ERD) with Cardinality using Crow’s Foot Notation:

|  |
| --- |
| **Doctor** |
| DoctorID (PK) |
| DoctorName |
| ContactNumber |

|  |
| --- |
| **ServicePoint** |
| ServicePointID (PK) |
| ServicePointName |

|  |
| --- |
| **Department** |
| DepartmentID (PK) |
| DepartmentName |
| **Doctor\_ServicePoint** |
| DoctorID (FK) |
| ServicePointID (FK) |

|  |
| --- |
| **Doctor\_Department** |
| DoctorID (FK) |
| DepartmentID (FK) |

This ERD displays the relationships between Doctors, Service Points, and Departments. The many-to-many relationships between doctors and their associated service points and departments are represented by the connecting tables (Doctor\_ServicePoint and Doctor\_Department).

**Ans to the Question—2**

Initially, n = 30.

- Iteration 1 (i = 0): `n += 0`, so `n = 30` (no change)

- Iteration 2 (i = 1): `n += 1`, so `n = 31`

- Iteration 3 (i = 2):`n += 2`, so `n = 33`

- Iteration 4 (i = 3): `n += 3`, so `n = 36`

- Iteration 5 (i = 4): `n += 4`, so `n = 40`

- Iteration 6 (i = 5): `n += 5`, so `n = 45`

The loop completes all 6 iterations (i goes from 0 to 5). The final value of `n` after the loop will be 45.

**Ans to the Question—4**

explain method overloading and method overriding in C# with examples.

Method Overloading:

Method overloading in C# allows a class to have multiple methods with the same name but with different parameters. The methods must have different parameter lists (different number or type of parameters). This enables the programmer to use the same method name for different behaviors.

Example of Method Overloading:

using System;

class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

public double Add(double a, double b)

{

return a + b;

}

}

class Program

{

static void Main()

{

Calculator calc = new Calculator();

int result1 = calc.Add(5, 10);

Console.WriteLine("Result 1: " + result1);

double result2 = calc.Add(3.5, 2.5);

Console.WriteLine("Result 2: " + result2);

}

}

```

In the example above, the Calculator class demonstrates method overloading with the Add method. It defines two Add methods with different parameter types (int and double). This allows the same method name to perform addition for both integers and doubles.

Method Overriding:

Method overriding in C# occurs when a child class provides a specific implementation of a method that is already defined in its parent class. It is commonly used in inheritance and allows a subclass to provide a specialized implementation of a method that is already defined in its superclass.

Example of Method Overriding:

using System;

class Animal

{

public virtual void MakeSound()

{

Console.WriteLine("Animal makes a sound");

}

}

class Dog : Animal

{

public override void MakeSound()

{

Console.WriteLine("Dog barks");

}

}

class Program

{

static void Main()

{

Animal animal = new Animal();

animal.MakeSound();

Dog dog = new Dog();

dog.MakeSound();

}

}

```

**Ans To The Question-5**

here is the translation of the UML class diagram into C# code:

using System;

public class Clinician

{

public string name { get; set; }

public string hospitalName { get; set; }

protected bool isSessionExists(string username)

{

return true;

}

}

public class Doctor : Clinician

{

public string practiceNumber { get; set; }

public void createPrescription(int patientNumber)

{

Console.WriteLine("Prescription created for patient number: " + patientNumber);

}

}

public class Pharmacist : Clinician

{

public string pharmacistNumber { get; set; }

public void dispenseMedications(int prescriptionNumber)

{

Console.WriteLine("Medications dispensed for prescription number: " + prescriptionNumber);

}

}

class Program

{

static void Main()

{

Doctor doctor = new Doctor();

doctor.name = "Dr. John";

doctor.hospitalName = "General Hospital";

doctor.practiceNumber = "12345";

Pharmacist pharmacist = new Pharmacist();

pharmacist.name = "Alice";

pharmacist.hospitalName = "Community Pharmacy";

pharmacist.pharmacistNumber = "54321";

doctor.createPrescription(1001);

pharmacist.dispenseMedications(5005);

}

}

```

The base class Clinician includes common attributes and a protected method for checking session existence. The subclasses Doctor and Pharmacist inherit from the Clinician class and add specific attributes and methods related to their roles (createPrescription for a doctor and dispenseMedications for a pharmacist). The Main method showcases how these methods can be used in a basic context.

**Ans To The Question-6**

using System;

class Program

{

static void Main()

{

Console.WriteLine("Enter the value for n1:");

int n1 = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the value for n2:");

int n2 = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the value for n3:");

int n3 = int.Parse(Console.ReadLine());

int min;

if (n1 < n2)

{

min = n1;

}

else

{

min = n2;

}

if (n3 < min)

{

min = n3;

}

Console.WriteLine("The minimum value among n1, n2, and n3 is: " + min);

}

}

```

This C# code represents the activity depicted in the UML diagram. It first prompts the user to enter three integer values for n1, n2, and n3. Then it evaluates the conditions to find the minimum among these three values and finally prints the minimum value.