Bijay Shrestha
Roll : 11
NCC

PAGE :
DATE : / /

#1 Write a program to create class Employee with following specifications :

a. Field Members : firstName, lastName, salary
b. Properties : FirstName, LastName, FullName, Salary
c. Parameterized Constructor
d. Method : IncrementSalary (double s), Display () Method for showing details of employee
e. Now create an object of this class & show use of
   i) Constructor
   ii) Display the Full Name & Salary using Properties
   iii) Change the First Name
   iv) Increment the Salary by 10%.
   v) Display Full Name and Salary by calling method.

→

```
class Employee
{
    privak string _firstName;
    private string _lastName;
    private double _salary;

    public string FirstName
    {
        get => _firstNare;
        set
        { _firstName = value; }
    }
```

```
public string lastName
{
    get => _lastName;
    set
    { _lastName = value; }
}

public double Salary
{
    get => _salary;
    set
    { _salary = value }
}

public Employee (string firstName, string lastName,
    double salary)
{
    _salary = salary;
    _lastName = lastName;
    _firstName = firstName;
}

public void IncrementSalary (double s)
{
    salary += s;
}

public void Display ()
{
    Console.WriteLine ("Full Name is {0} {1}",
                        _firstName, _lastName);
```

```
    Console.WriteLine ("Salary is {0}", -salary);
  }

}


static void Main (string[] args)
{
    Employee e1 = new Employee ("Bijaya",
                                "Shrestha", 10000);
    Console.WriteLine ("Full Name is {0} {1}",
                        e1.FirstName, e1.LastName);
    Console.WriteLine ("Salary is {0}", e1.Salary);
    e1.FirstName = "Bijay";

    Console.WriteLine ("First Name is changed as {0}",
                        e1.FirstName);
    double incSal = 0.1 * e1.Salary;
    e1.IncrementSalary (incSal);
    Console.WriteLine ("After increment by 10% salary is {0}",
                        e1.Salary);

    e1.Display ()
    Console.ReadLine ();
}
```

*** Output ***

```
Full Name is Bijaya Shrestha
Salary is 10000
First Name is changed as Bijay
After increment by 10% salary is 11000
```

```
Full Name is Bijay
        Shrestha
Salary is 11000
```

PAGE :
DATE : / /

#2 What is Polymorphism ? Explain the different type of approaches to implement the polymorphism with example.

→ The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It occurs when we have many classes that are related to each other by inheritance. Simply, it is the ability of different objects to respond in a unique way to the same message.

The different type of approaches to implement the polymorphism are :

i) Static / Compile Time Polymorphism
It is also known as Early Binding. Method overloading is an example of Static Polymorphism. It is also called as Compile Time Polymorphism because the decision of which method is to be called is made at compile time.
In the following example, a class has two methods with the same 'Add' but with different input parameters (the first has three parameters and the second method has two parameters).

```
static public int Add (int a, int b, int c) => a+b+c;
static public int Add (int a, int b) => a+b;
static void Main (string[] args) {
    Console.WriteLine ($" Add 3 items gives {Add (5,9,3)}");
    Console.WriteLine ($" Add 2 items gives {Add (5,7)}");
}
```

ii) Dynamic / Runtime Polymorphism

It is also known as late binding. Method overriding is an example of dynamic polymorphism. The compiler will decide which method to call at runtime and if no method is found then it throws an error.

Example :-

```
public class Animal
{
    public string color = "white";
    public virtual void eat()
    {
        Console.WriteLine("Eating");
    }
}
public class Dog : Animal
{
    public override void eat()
    {
        Console.WriteLine("Dog is Eating");
    }
}
static void Main(string[] args)
{
    Animal a = new Dog();
    a.eat();
    Console.WriteLine($"Color is {a.color}");
}
```

#3 How do you create and implement interfaces to achieve multiple inheritance in C# ? Explain with suitable example.

→ In Multiple inheritance, one class can have more than one superclass and inherit features from all its parent classes. As shown in the below diagram, class C inherits the features of class A and B

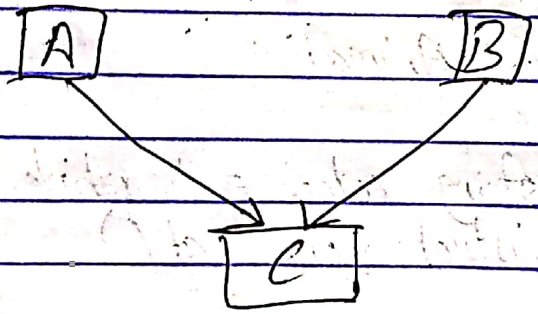
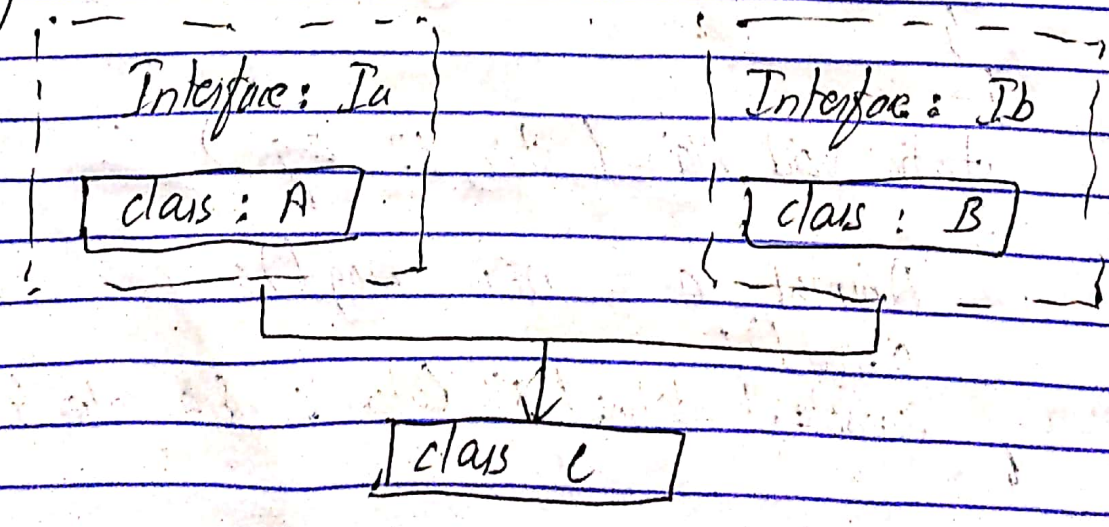


fig :- multiple inheritance

But C# does not support multiple class inheritance. To overcome this problem, we can create and implement interfaces to achieve multiple class inheritance. With the help of the interface, class C can get the features of class A and B as shown in fig below :-



PAGE:
DATE: / /

Simple example to achieve multiple inheritance in C# using interface is

```
public interface Ia
{ void f1(); }

public class A : Ia
{
    public void f1()
    {
        Console.WriteLine("f1 in A");
    }
}
```

```
public interface Ib
{ void f2(); }

public class B : Ib
{
    public void f2()
    {
        Console.WriteLine("f2 in B");
    }
}
```

```
public class C : Ia, Ib
{
    A a = new A();
    B b = new B();

    public void f1()
    { a.f1(); }
    public void f2()
    { b.f2(); }
}

static void Main(string[] args)
{
    C c = new C();
    c.f1();
    c.f2();
}
```

PAGE
DATE : / /

#4 What is collection ? Explain with example.

→ In C# collection is specialized classes for data storage and retrieval. It represent group of objects. Collection classes seive various purpose, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. We can perform various operations on objects such as

- Store
- Update
- Delete
- Retrieve
- Search
- Sort

The namespaces for using collection with their classes in C# are

i) System.Collection
It includes, ArrayList, Stack, Queue, HashTable, SortedList.

ii) System.Collection.Generic
It includes List, Dictionary.

An example of collection :-

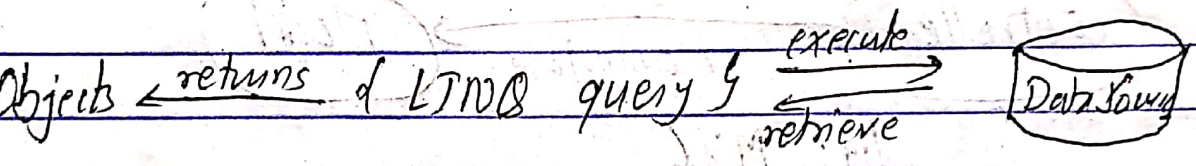
Use of List

```
List<string> names = new List<string>();
names.Add("Ram");
names.Add("Hari");
names.Add("Sam");
Console.WriteLine(names.Count);
names.RemoveAt(2);
foreach (string name in names)
    Console.WriteLine(name);
```

PAGE:
DATE: / /

#5 What is LINQ ? Explain LINQ method syntax and give an example ?

→ Language - Integrated Query (LINQ) is the name for a set of technologies based on the integration of query capabilities directly into the C# language. It is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats such as collections, ADO.NET Dataset, XML Docs, web Servie and MS SQL Servs and othes databases.



For an example :-

```
// Data source
string [] names = { "Bill", "Steve", "James", "Mohan" };

// LINQ Query
var myLinqQuery = from name in names
                  where name.Contains ('a')
                  select name;

// Query execution
foreach (var name in myLinqQuery)
    Console.Write (name + " ");
```

In the above exampk, string array names is a data source. In a variable myLinqQuery, a LINQ Query is assigned, which gets all the strings from an array which contains 'a'. Finally, a foreach loop is used to output the ~~name~~ results.

#6 Explain the Model, Controller and View in MVC.

→ MVC stands for Model, View, and Controller. It separates an application into three components- Model, View and Controller.

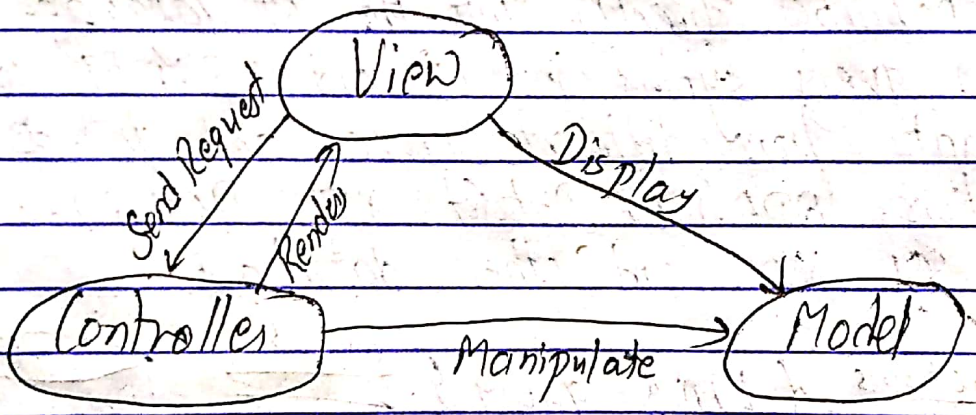


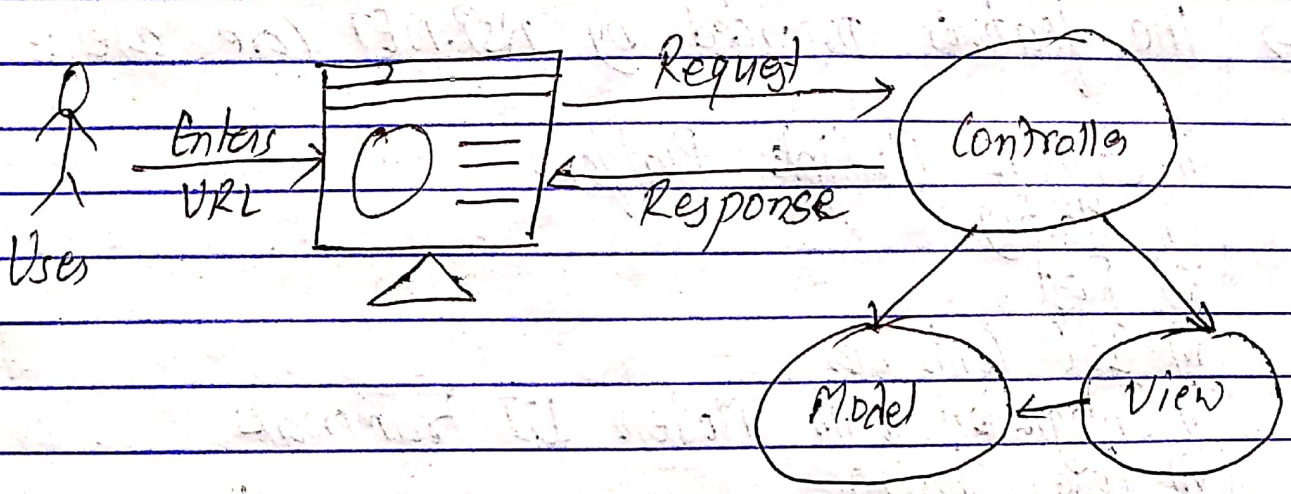
fig: MVC

Model :-
It represents the shape of the data. A class in C# is used to describe a model. Model objects store data retrieved from the database. Model represents the data.

View :-
It is a user interface. View display model data to the user and also enables them to modify them. View in ASP.NET MVC is HTML, CSS, and some special syntax (Razor syntax) that makes it easy to communicate with the model and the controller.

Controller :-
It handles the user request. Typically, the user uses the views and raise an HTTP request.

Controller processes request and returns the appropriate view as a response. It is the request handler.



Q7 Define ASP.NET Web API and write its characteristics.

→ ASP.NET Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Framework. It only support HTTP protocol.

It's characteristics are :-

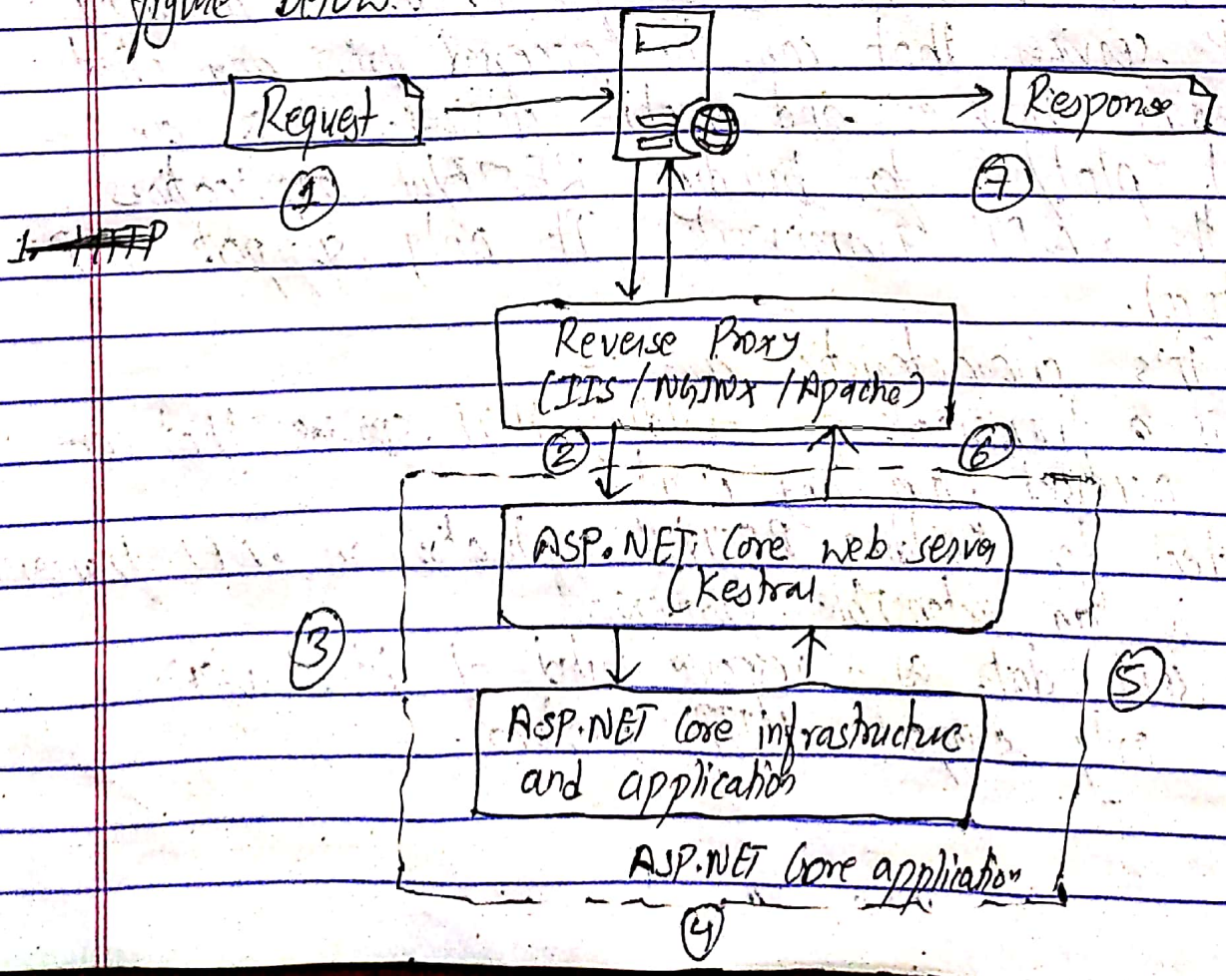
i) It is a framework for building HTTP services that can be accessed from any client.
ii) Ideal for building RESTful applications on .NET framework.
iii) It is an extensible framework
iv) It sends data as a response instead of html view.
v) It only supports HTTP protocol.

#8 - What are the features provided by ASP.NET Core? How does ASP.NET Core proces a request?

→ The features provided by ASP.NET Core are :-

i) Support Multiple Platforms
ii) Hosting
iii) Fast
iv) IoC Containes
v) Integration with Modern UI Frameworks
vi) Code Sharing
vii) Side-by-Side App Versioning
viii) Smalles Deployment Footprint.

ASP.NET Core proces a request as shown in figure below :-



PAGE:
DATE: / /

1.) HTTP request is made to the server and is received by the reverse proxy

2.) Request is forwarded by IIS / NGINX / Apache to ASP.NET Core

3.) ASP.NET Core server receive the HTTP request and passes it to the middleware

4.) Request is processed by the application, which generates the response.

5) Response passes through middleware back to web server.

6) Web server forwards response to reverse proxy.

7) HTTP response is sent to browser.

PAGE :
DATE / /

#9 Explain the common web application architecture.

→ The common web application architecture are explained below :-

i) Monolithic application
- It is one that is entirely self-contained, in terms of its behaviour
- It may interact with other services or data stores in the course of performing its operations, but the core of its behaviour runs within its own process and the entire application is typically deployed as a single unit.
- If such an application needs to scale horizontally, typically the entire application is duplicated across multiple servers or virtual machines.

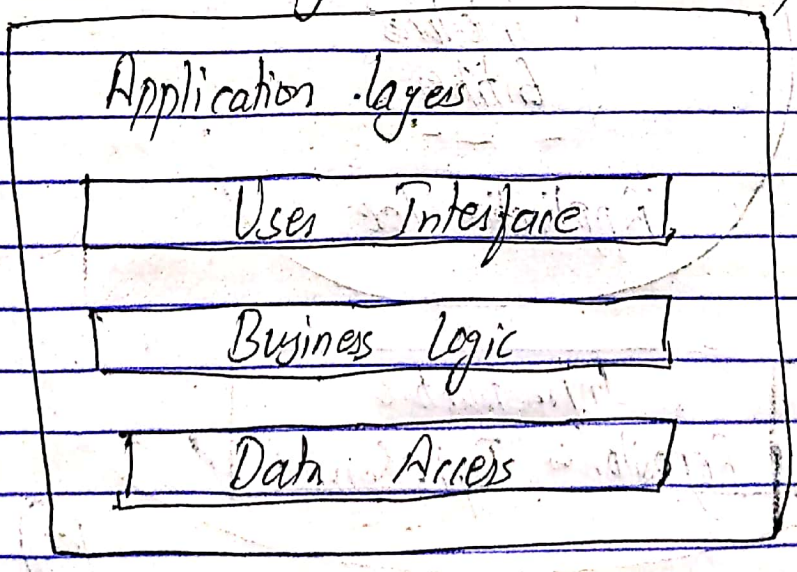
ii) All-in-one applications.
- The smallest possible number of projects for an application architecture is one. In this architecture, the entire logic of the application is contained in a single project, compiled to a single assembly, and deployed as a single unit.
- It contains all of the behavior of the application, including presentation, business, and data access logic.
- In a single project scenario, separation of concerns is achieved through the use of folders.
- Presentation details should be limited as much as possible to the Views folder, and data access implementation details should be limited to classes kept in the Data folder. Business logic should reside in services and classes within the Models folder.

PAGE:
DATE: / /

iii) Layered Architecture

- It offers a number of advantages beyond just code organization, though. By organizing code into layers, common low-level functionality can be reused throughout the application.
- It helps to achieve encapsulation.
- When a layer is changed or replaced, only those layers that work with it should be impacted.
- By limiting which layers depend on which other layers, the impact of changes can be mitigated so that a single change doesn't impact the entire application.

iv) Traditional "N-layer" architecture applications



- These layers are frequently abbreviated as UI, BLL and DAL.
- Using this architecture, users make requests through the UI layer, which interacts only with the BLL.
- The BLL, in turn, can call the DAL for data access requests.
- The UI layer shouldn't make any requests to the DAL

directly, nor should it interact with persistence directly through other means.
- Likewise, the BLL should only interact with persistance by going through the DAL.

v) Clean architecture

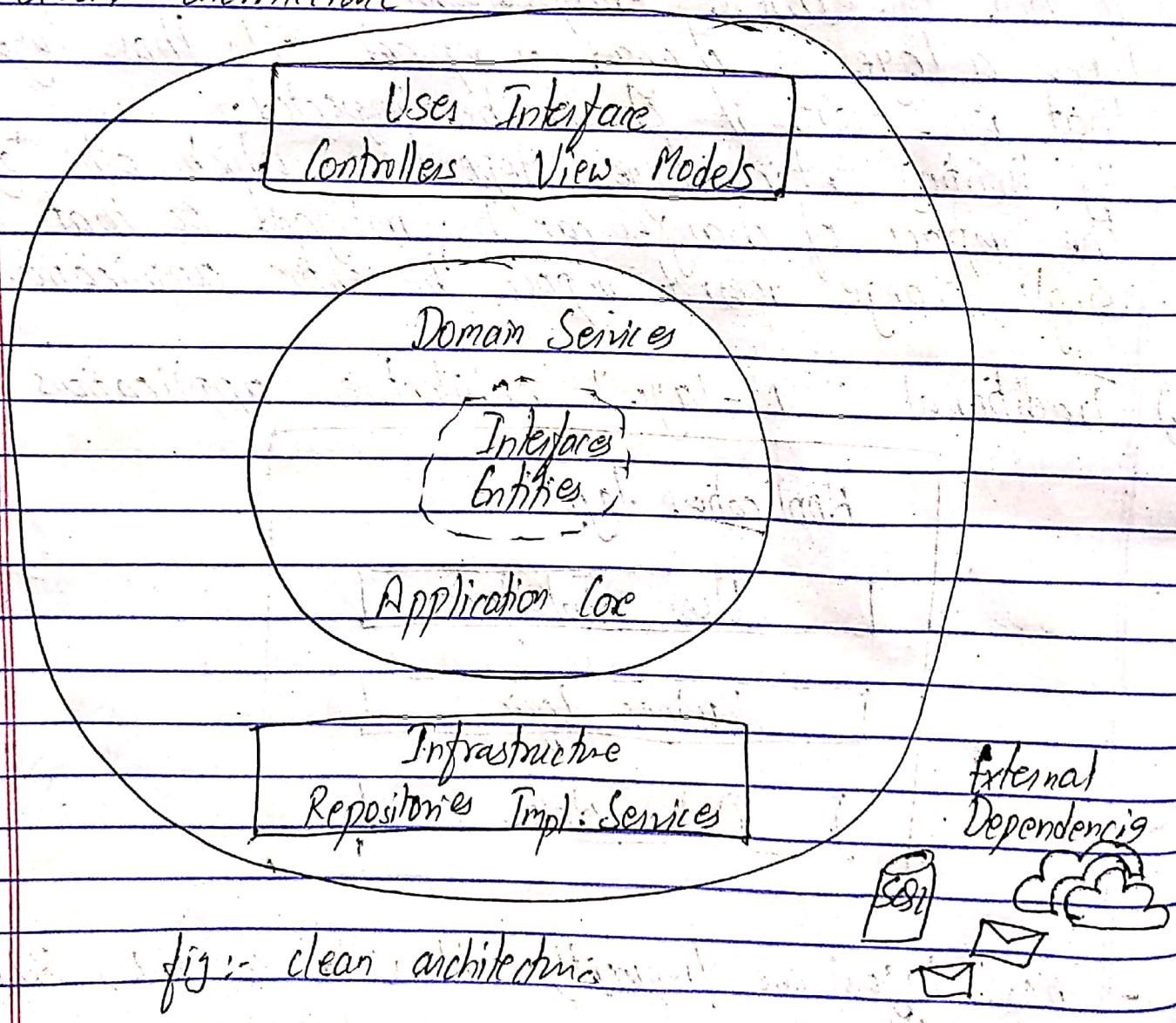


fig:- clean architecture

- Applications that follow the Dependency Inversion Principle as well as the Domain-Driven Design (DDD) principles tend to arrive at a similar architecture. It's been cited as the Onion Architecture or Clean Architecture.

PAGE :
DATE : / /

- Clean architecture puts the buyiness logic and application model at the centre of the application. Instead of having buyiness logic depend on data access or other infrastructure concerns, this dependency is inverted: infrastructure and implementation details depend on the Application Core.
- This is achieved by defining abstractions, or interfaces, in the Application Core, which are then implemented by types defined in the Infrastructure layer. A common way of visualizing this architecture is to use a series of concentric circles, similar to an onion