DAY: _____ DATE: _____

EAD-01 (20/01/25)

⇒ Framework

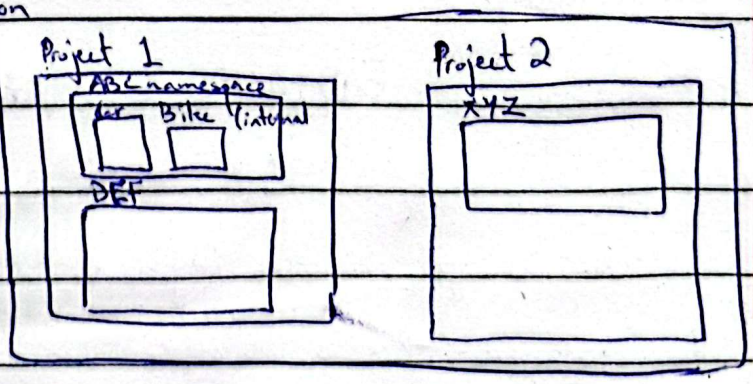
↳ provide facility.

↳ Already written code.

EAD-02 (23/01/25)

dll files → own library files/packages.

Solution



⇒ One sol can have d/f projects.

1) Desktop 2) Mobile ...

Same Project

↳ Car can have Book, Bike.

DEF ⇒ using ABC

↳ can access Car + Bike (internal)

Diff Project

XYZ ⇒ using ABC

↳ only public classes ✓

internal classes ✗

↳ WPF

↳ Entity framework for DB queries.

1) Desktop 2) Web 3) APIs.

Projects:

⇒ Yt ⇒ Attendence ⇒ Quiz.

EAD-03 (27/01/25)

C#, All Plat, Console

.net → windows only.

.net-core → cross platform (mac+linux also)

Folder Structure:

↳ Solution [project [namespace [class,...],...],...]

DATE: _____ DATE: _____

↳ A project req 1 file with atleast 1 namespace & 1 public class containing public static void Main()

↳ If same name of namespace in d/f files, we can access classes from other files (as they are in same file)

↳ If 2 namespace in single file, cannot access other namespace's classes

↳ using <namespace>

↳ **Classes Access Modifier:**

(default) 1) Public → other namespace + projects.

2) Internal → other namespace with same project.

3) Private → same namespace only.

↳ Class members are default private.

library ⇒ dll files (binary).

↳ no output

↳ classes + methods.

↳ Add dll files to dependencies.

Project 2 needs project 1 reference to access public classes.

DATE: DATE:

Class Member Access Modifiers:

1) Public → Other ,
2) Private →
3) Internal →

Exe → 1) 2 project link.
2) External library link
3) nuget package.

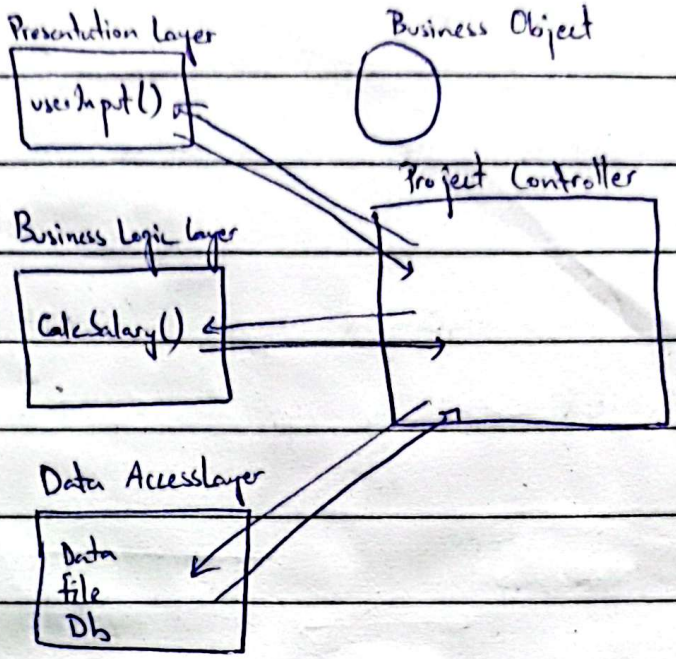
```
// no of parameters in func not fixed.
public int sum(params int [] elements)
{ int sum = 0;
    foreach (int ele in elements)
        sum += ele;
    return sum;
}
```

⇒ System + some other namespaces included by default.

DATE: _____ DATE: _____

EAD-04 (30/01/25)

N-tier Architecture:



⇒ If Db needs to be changed only Data Access Layer will be changed.

⇒ Desktop to Mobile ⇒ Presentation Layer.

1) Add Project (library - PresentationLayer)
2) ~~~~ BusinessLogicLayer)
3) ~~~~ DataAccessLayer)
4) ~~~~ BusinessObject(Layer)

⇒ Add project ref & namespaces to mainController & then use classes.

⇒ getter/setter ⇒ imposing logic like age > 10

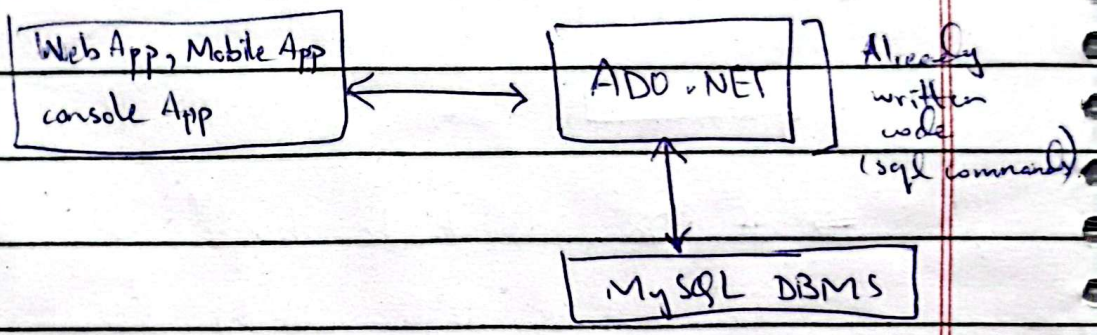
⇒ Emp emp1 = new Emp {Age = 15, ...}

PAGE: _____ DATE: _____

```
⇒ private string str_name;
  public string Str_name{
    get { return str_name; }
    set { str_name = value; }
  }
```

EAD-05 (03/02/25)

ADO.NET → connects Application & Data Source.



CRUD

SQL Injection

↳ Parameterized Query (Solution)

↳ Entity Framework.

⇒ Microsoft SQL Client Package.

→ SQL Server Object Explorer.

⇒ IDENTITY PRIMARY KEY.

↳ Auto key + increment.

⇒ Select Query → ExecuteReader()

→ Insert, Update, Del → ExecuteNonQuery().

DAY: _____ DATE: _____

~~ for commenting queries.

EAD-06 (06/02/25)

→ Sol of SQL Injection.

↳ Parameterized queries:

~~ where username = '@abc' ~~

SqlParameter p1 = new SqlParameter("abc", uname);

cmd.Parameters.Add(p1);

1- Create connection (conn string).

2- create command (query, conn)

3- Add Parameters to command

4- Execute query.

⇒ use try catch.

// some objects need to be deleted from RAM ASAP

↳ using (Program p = new Program())
{ //use here }

↳ require IDisposable interface to be implemented.

class car() : IDisposable {
public void Dispose() {
} }

DATE: _____ DATE: _____

EAD-07 (17/02/25)

⇒ WPF Application Project

⇒ backend (C#) , frontend (XML)

↳ Drag Drop (not responsive), xcode

⇒ 4 Panels for frontend:

1) dock 2) stack 3) wrap 4) grid.

Stack Panel: (single row / col)

↳ Orientation ⇒ default vertical.

↳ nested panels possible

Dock Panel:

↳ When we want to dock element e.g top

↳ dock = "left" (default).

↳ lastChildFill = "True"

↳ takes remaining space.

Grid Panel:

↳ Divides the UI into rows & cols.

↳ Wrap Panel:

↳ Similar to StackPanel, but wraps items to the next row/column when there's no space.

EAD-08 (19/02/25)

NuGet Package Tools ⇒ (Package Manager Installation)

1) Microsoft.EntityFrameworkCore.SqlServer
2) ——————————————.Design
3) ——————————————.Tools

↳ Tables should already exist in db.

↳ makes class from table.

(Table) users ⇒ (Class) User

↳ DbSet <User> users;

⇒ Package Manager Console.

↳ // for db Connection.

SqlContext db = new SqlContext();

List<Vetuser> lst = db.Vetusers.toList();

DbSet (table) ↲

```
using (SqlContext db = new SqlContext()){
    Vetuser tmp = new VetUser{username = "VET",
                              password = "KSK123"};
    db.Vetusers.Add(tmp);
    db.Save Changes();
}
```

EAD-09 (24/02/25)

↳ Class first Approach

↳ DB first Approach

⇒ Entity framework makes classes from tables & list of objects in class of DB_NAME.

```
Se2lContext db = new Se2lContext();
User obj;                          //only ref var is made.

// get list    List<VetUser> lst = db.Vetusers.ToList();
// insert obj  User tmp = new ___
               db.VetUser.Add(tmp)
               db.SaveChanges();
```

↳ To sync RAM changes with DB.

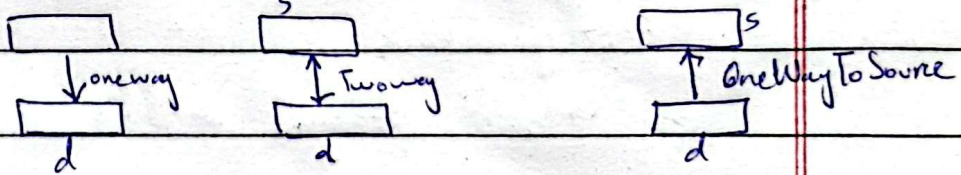
DAY: _____ DATE: _____

// delete obj; var obj = db.VetUsers.FirstOrDefault
(x => x.username == "VET")
db.VetUsers.Remove(obj);
db.SaveChanges();

// update obj;
// get // edit // saveChanges();

Imp **Data Binding:**

1) Assign names 2) Remove text property.
3) Text = "{Binding ElementName = source
Path = Text Mode = OneWay}



⇒ Two way mode require
UpdateSourceTrigger = PropertyChanged.
(to update automatically).

Example:

TextBox ☐ checkbox.

without binding {
if (textbox.text == "check")
check1.isChecked = true.

DATE: _____ DATE: _____

↳ make Convertor class.

TextToBool.

<WindowResources>

<local:TextToBool x:key="myConvertor">

</local:TextToBool>

</WindowResources>

⇒ isChecked = "{Binding ElementName=TY-source

Path=Text

Convertor={StaticResource myConvertor}}"

Exercise:

↳ convertback.

↳ bind datagrid (edit/delete).

{

DATE: DATE:

EAD-10 (27/02/25)

List <Product> lst = new List<Product>();

↳ List contains list of addresses.

DataGrid Binding

⇒ mygrid.

<DataGrid Name="mygrid" AutoGenerateColumns="False" CanUserAddRows="False">

<DataGrid.Columns>

<DataGridTextColumn Header="__" width="A" Binding={Binding pName}

</Data

<DataGridTemplateColumn

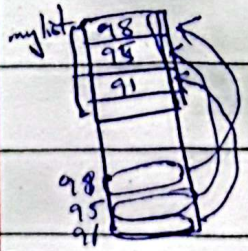
↳ frontend will not auto update

list does not handle inotify

list ⇒ ObservableCollection.

// Product tmp= (mygrid.selectedRow as Product)

lst.remove(tmp).



mylist ⇒ Link list.

DATE: _____ DATE: _____

EAD-11 (3/3/25)

try{}catch(Exception e){ }
↳ will avoid app to crash.

// Edit/Update

⇒ when edit (any) button is clicked in grid it becomes selected.

⇒ We can get address of selected row as
Product temp = (mygrid.SelectedItem as Product);

⇒ When obj is added / deleted in list, it notifies grid but when existing obj is changed it does not notifies list

⇒ class Product: INotifyPropertyChanged {

private int price;

public int Price { get { return price; }
set { price = value;
Notify("Price");
}
}