Name : HEMANT SOPAN PATIL

Class : MSC ICT 2

Practical Assignment – 2

SUBJECT :- .NET CORE

1)Create a C# application to demonstrate use of Generic Abstract class.

Code :-

Generic Abstract Class :-

```
abstract class Account<T>
{
    public T AccountNumber;
    public double Balance;

    1 reference
    public Account(T accNo, double balance)
    {
        AccountNumber = accNo;
        Balance = balance;
    }

    2 references
    public abstract void Deposit(double amount);
    2 references
    public abstract void Withdraw(double amount);

    1 reference
    public void Display()
    {
        Console.WriteLine("Account Number: " + AccountNumber);
        Console.WriteLine("Balance: " + Balance);
    }
}
```

Derived Class :-

```
3 references
class SavingsAccount : Account<int>
{
    public string HolderName;

    1 reference
    public SavingsAccount(int accNo, double balance, string name)
        : base(accNo, balance)
    {
        HolderName = name;
    }

    2 references
    public override void Deposit(double amount)
    {
        Balance = Balance + amount;
        Console.WriteLine("Deposited Amount: " + amount);
    }

    2 references
    public override void Withdraw(double amount)
    {
        if (amount <= Balance)
        {
            Balance = Balance - amount;
            Console.WriteLine("Withdrawn Amount: " + amount);
        }
        else
        {
            Console.WriteLine("Not enough balance");
        }
    }

    1 reference
    public void ShowDetails()
    {
        Console.WriteLine("Account Holder: " + HolderName);
        Display();
    }
}
```

Output :-

```
Deposited Amount: 2000
Withdrawn Amount: 1500
Account Holder: Hemant
Account Number: 101
Balance: 5500
```

2) Create an MVC Core web application to perform CRUD operations for Project allocation to employees using EntityFramework. Also create advance search functionality to search employee on project.

Code :-

Employee class :-

```
namespace EmployeePortal.Models
{
    13 references
    public class Employee
    {
        11 references
        public int Id { get; set; }

        13 references
        public string Name { get; set; }

        13 references
        public string Project { get; set; }

        13 references
        public string Deparment { get; set; }

        12 references
        public DateOnly Joiningdate { get; set; }
    }
}
```

ApplicationDBContext :-

Insert :-

```
public IActionResult Create()
{
    return View();
}

// POST: Employees/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("Id,Name,Project,Deparment,Joiningdate")] Employee employee)
{
    if (ModelState.IsValid)
    {
        _context.Add(employee);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(employee);
}
```

Delete :-

```
    // GET: Employees/Delete/5
    0 references
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var employee = await _context.Employees
            .FirstOrDefaultAsync(m => m.Id == id);
        if (employee == null)
        {
            return NotFound();
        }

        return View(employee);
    }

    // POST: Employees/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    0 references
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var employee = await _context.Employees.FindAsync(id);
        if (employee != null)
        {
            _context.Employees.Remove(employee);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
```

Update :-

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Project,Deparment,Joiningdate")] Employee employee)
{
    if (id != employee.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(employee);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!EmployeeExists(employee.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(employee);
}
```

Output :-

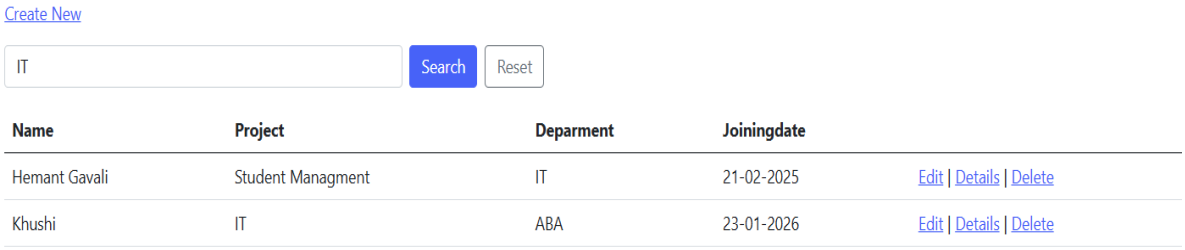
Name	Project	Deparment	Joiningdate	
Hemant Gavali	Student Managment	IT	21-02-2025	Edit | Details | Delete
Hemant Patil	Cafe Managment	Tech	21-02-2024	Edit | Details | Delete
Khushi	IT	ABA	23-01-2026	Edit | Details | Delete

Search Implementation :-

```
public async Task<IActionResult> Index(string searchText)
{
    var employees = _context.Employees.AsQueryable();

    if (!string.IsNullOrEmpty(searchText))
    {
        employees = employees.Where(e =>
            e.Name.Contains(searchText) ||
            e.Deparment.Contains(searchText) ||
            e.Project.Contains(searchText)
        );
    }

    return View(await employees.ToListAsync());
}
```

```
<form method="get" asp-action="Index" class="mb-3">
    <div class="row g-2 align-items-center">
        <div class="col-md-4 col-sm-6">
            <input type="text"
                   name="searchText"
                   class="form-control"
                   value="@Context.Request.Query["searchText"]" />
        </div>

        <div class="col-auto">
            <button class="btn btn-primary" type="submit">
                Search
            </button>
        </div>

        <div class="col-auto">
            <a asp-action="Index" class="btn btn-outline-secondary">
                Reset
            </a>
        </div>
    </div>
</form>
```

Output :-



3) Demonstrate use of String Indexer and multivalued Indexer to search data from a Generic collection.

Code :-

class with indexers :-

```
2 references
class Library
{
    List<Book> books = new List<Book>();

    3 references
    public void AddBook(Book b)
    {
        books.Add(b);
    }

    1 reference
    public Book this[string title]
    {
        get
        {
            foreach (Book b in books)
            {
                if (b.Title == title)
                {
                    return b;
                }
            }
            return null;
        }
    }
```

Multivalued Indexer :-

```
public Book this[int id, string author]
{
    get
    {
        foreach (Book b in books)
        {
            if (b.Id == id && b.Author == author)
            {
                return b;
            }
        }
        return null;
    }
}
```

Main Class :-

```
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        Library lib = new Library();

        lib.AddBook(new Book(1, ".Net", "Hemant"));
        lib.AddBook(new Book(2, "Java", "Rohit"));
        lib.AddBook(new Book(3, "Python", "Anita"));

        Book b1 = lib[".Net"];
        Console.WriteLine("Search using String indexer:");
        Console.WriteLine(b1.Title + " - " + b1.Author);

        Book b2 = lib[2, "Rohit"];
        Console.WriteLine("Search using multivalued Indexer :");
        Console.WriteLine(b2.Title + " - " + b2.Author);

        Console.ReadLine();
    }
}
```

Output :-

```
Search using String indexer:
.Net - Hemant
Search using multivalued Indexer :
Java - Rohit
```