1. Demonstration of Constructors : Define a class Student, which contains the following information about students: Full Name, Semester, Course_Code, Subject, e-mail and phone number. Declare several constructors for the class Student, which have different lists of parameters (for complete information about a student or part of it). Data, which has no initial value to be initialized with null. Add a method in the class Student, which displays complete information about the student.

```
using System ;
class Student
{
public string FullName { get; set; }
public int Semester {get; set; }
public string CourseCode { get; set; }
public string Subject{ get; set; }
public string Email { get; set; }
public string PhoneNumber { get; set; }

public Student()
{

}
public Student(string fullNams, int semester, string courseCode, string subject, string email,
string phoneNumber)
{
FullName = fullName;
Semester = semester;
CourseCode = courseCode;
Subject= subject;
Email =email;
PhoneNumber = phoneNumber;
}
public Student(string fullNams, int semester, string courseCode)
{
FullName = fullName;
Semester = semester;
CourseCode = courseCode;
Subject= null;
Email = null;
PhoneNumber = null;
}
// Method to display complete information
public void Display(information()
{
Console.WriteLine("Full Name: " + FullName);
Console.WriteLine("Semester: " + Semester) ;

Console.WriteLine("Course Code: " + CourseCode);
Console.WriteLine("Subject: " + (Subject ?? "N/A");
Console.WriteLine("Email: " + (Email ?? "N/A");
Console.WriteLine("Phone Number: " + (PhoneNumber ?? "N/A");
}
}
class Program
{
static void Main(string[] args)
{
// Using different constructors
Student student1 = new Student("John", 3, "CS101", "Dot Net Framework",
"john@example.com", "123-456-7890");
Student student2 = new Student("Joseph", 2, "ENG201");
Student student3 = new Student();
// Displaying information
Console.WriteLine("Student 1 information:");
student1.Display(Information();
Console.WriteLine();
Console.WriteLine("Student 2 information:");
student2. Display(Information();
Console.WriteLine();
Console.WriteLine("Student 3 information:");
student3.Display(Information();
}
}
```

2.Demonstration of Delegates : Create a delegate called strMyDel that takes one string parameter and returns a string. Create a class named TestDelegate that contains two non-static methods changecase() and reverse() having following signature: String changecase(String str); Changes the case of input characters , String Reverse(String str); - reverse the given string Use multicast delegate

```
using System ;
using System .Collections.Generic;
delegate string StrDel(string str);
class TestDelegate
{
// Define the multicast delegate
private StrDel strMyDel;
// Constructor
public TestDelegate()
{
strMyDel = new StrDel(ChangeCase);
strMyDel += Reverse;
}
// Method to change case of input characters
private string ChangeCase(string str)
{
char[] arr = str.ToCharArray();
for (int i = 0; i < arr.Length; i++)
{
if (Char.IsLower(arr[i]))
arr[i] = Char.ToUpper(arr[i]);
else
arr[i] = Char.ToLower(arr[i]);
}
return new string(arr);
}
// Method to reverse the given string
private string Reverse(string str)
{
char[] arr = str.ToCharArray();
Array.Reverse(arr);
return new string(arr);
}
// Method to invoke the delegate
public string InvokeDelegate(string str)
{
return strMyDel(str);
}
}
class Program
{
static void Main(string[] args)
{
TestDelegate obj = new TestDelegate();

string input= "Hello World!";
Console.WriteLine("Original String: " + input);
Console.WriteLine("Changed Case: " + obj.InvokeDelegate(input));
Console.ReadKey();
}
}
```

3.Demonstration of Exceptions: Write a method ReadNumber(int start, int end) that reads an integer from the console in the range [start...end]. In case the input integer is not valid or it is not in the required range throw appropriate exception. Using this method, write a program that takes 10 integers a1, a2,. a10 such that 1 < a1 < ... < a10 < 100

```
using System ;
class Program
{
static void Main()
{
int[] numbers = new int[10];
for (int i = 0; i < 10; i++)
{
numbers[i] = ReadNumber(1, 100);
}
Console.WriteLine("Numbers entered:");
foreach (int number in numbers)
{
Console.WriteLine(number);
}
}
static int ReadNumber(int start, int end)
{
Console.Write("Enter a number between {0} and {1}: ", start, end);
string input= Console.ReadLine();
if (!int.TryParse(input, out int number))
{
throw new Exception("Invalid input. Please enter a valid number.");
}
if (number < start || number > end)
{
throw new Exception("Number is not within the specified range.");
}
return number;
}
}
```

4.Explain the concept of aggregation in object-oriented programming using the given C# code as an example. Describe how the classes Address and Employee demonstrate aggregation, and provide an explanation of the relationship between them. Additionally, discuss the significance of aggregation in building modular and reusable code.

```
using System;
public class Address
{

public string addressLine, city, state;
public Address(string addressLine, string city, string state)
{
this.addressLine = addressLine;
this.city = city;
this.state = state;
}
}
public class Employee
{
public int id;
public string name;
public Address adrobj; //Employee HAS-A Address
public Employee(int id, string name, Address address)
{
this.id = id;
this.name = name;
this.adrobj = address;
}
public void display()
{
Console.WriteLine(id + " " + name + " " + adrobj.addressLine + " " + adrobj.city + " " +
adrobj.state);
}
}
public class TestAggregation
{
public static void Main(string[] args)
{
Address a1=new Address("G-13, Sec-3","Noida","UP");

Employee e1 = new Employee(1,"Sudharsan",a1);
e1.display();
}
}
```