PARTE 2: Ejercicio de patrones y antipatrones
Duración: 120 minutos

EJERCICIO 1 (30')

1- Analiza el siguiente código e identifica que antipatrón se está siguiendo.
2- A sabiendas de que nuevos métodos de cálculos de sueldo se podrán agregar en un corto plazo…
Explique y justifique qué patrón(es) resuelve(en) mejor el problema
   a. Muestre un diagrama de clases con la solución
   b. Realice los cambios en el código

```
class Program
        {
        static void Main(string[] args)
        {
            var employeeManager = new EmployeeManager();

            employeeManager.AddEmployee(new Employee { Name = "Lala",
HoursWorked = 40, HourlyRate = 25 });
            employeeManager.AddEmployee(new Employee { Name = "Pepe",
HoursWorked = 50, HourlyRate = 20 });

            Console.WriteLine("Total Payroll: $" +
employeeManager.CalculateTotalPayroll());

                  //  Comentario: 2023-07-05; Para qué está esto? No sé,
pero no lo saco por si algo se rompe...
            employeeManager.OldPayrollSystem();
                  employeeManager.OtherPayrollCalculation();

                employeeManager.AddEmployee(new Employee { Name = "Boss",
HoursWorked = 5, HourlyRate = 200 });
        }
    }

    public class Employee
    {
        public string Name { get; set; }
        public int HoursWorked { get; set; }
        public double HourlyRate { get; set; }
    }
```

```
public class EmployeeManager
    {
        private List<Employee> employees = new List<Employee>();
        private List<int> oldPayrollSystemData = new List<int>(); // ¿Qué
es esto?

        public void AddEmployee(Employee employee)
        {
            employees.Add(employee);
        }

        public double CalculateTotalPayroll()
        {
            double total = 0;
            foreach (var employee in employees)
            {
                total += employee.HoursWorked * employee.HourlyRate;
            }
            return total;
        }

        // Comentario: 2023-07-05;
            // Esto parece ser parte de un viejo sistema de nómina que ya
no se usa.
        // Pero está aquí, y no estamos seguros de si es seguro eliminarlo.
        public void OldPayrollSystem()
        {
           ...
           Console.WriteLine("Old payroll system processed.");
        }

        // Comentario: 2023-07-05;
            // Esto parece ser parte de un viejo sistema de nómina que ya
no se usa.
        // Pero está aquí, y no estamos seguros de si es seguro eliminarlo.
        public void OtherPayrollCalculation()
        {
           ...
           Console.WriteLine("Old payroll system processed.");
        }
    }
```

EJERCICIO 2 (30')

Se quiere implementar un módulo para gestión de usuarios.
Durante el piloto se realizó la siguiente prueba y el líder de equipo le encomendó la tarea de analizar el código y realizar los cambios necesarios para que sea más manejable cuando UserProfile crezca. En el marco de que podrá tener muchos cambios y del análisis del ambiente de trabajo se detectó que los operadores del sistema son muy propensos a cometer errores.

Explique y justifique qué patrón(es) resuelve(en) mejor el problema

1- Muestre un diagrama de clases con la solución
2- Realice los cambios en el código

```
// UserProfile.cs
public class UserProfile
{
    public string Name { get; set; }
    public int Age { get; set; }

    public UserProfile(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public void PrintProfile()
    {
        Console.WriteLine($"Name: {Name}, Age: {Age}");
    }
}

// Program.cs
class Program
{
    static void Main()
    {
        UserProfile profile = new UserProfile("Alice", 25);

        Console.WriteLine("Original Profile:");
        profile.PrintProfile();

        Console.WriteLine("\nUpdated Profile:");
        profile.Name = "Bob";
        profile.Age = 30;
        profile.PrintProfile();

        Console.WriteLine("\nUPS!!!:");
        profile.Name = "Alice";
        profile.Age = 25;
        profile.PrintProfile();
    }
}
```

EJERCICIO 3 (30')

El programa a continuación tiene una clase EmailNotificationSystem que envía notificaciones por correo electrónico y una clase SMSNotificationSystem que envía notificaciones SMS. Los usuarios pueden suscribirse o darse de baja de las notificaciones, y cuando se envía una notificación, se realiza a través de ambos sistemas.

Sin embargo, la clase EmailNotificationSystem y SMSNotificationSystem tienen interfaces diferentes y no están bien integradas con el resto del sistema. Además, el proceso de envío de notificaciones no está bien desacoplado.

1. Debe refactorear el código para permitir que EmailNotificationSystem y SMSNotificationSystem sean utilizados de manera intercambiable en el sistema.
2. Además, deberán desacoplar el proceso de envío de notificaciones y permitir que los usuarios se suscriban y reciban notificaciones de forma automática cuando se envíe una nueva notificación.

Explique y justifique qué patrón(es) resuelve(en) mejor el problema.
Muestre un diagrama de clases con la solución

```
class Program
    {
        static void Main(string[] args)
        {
            var emailSystem = new EmailNotificationSystem();
            var smsSystem = new SMSNotificationSystem();

            var message = "This is a notification message.";

            // Simulate subscribing users
            emailSystem.Subscribe("user1@example.com");
            smsSystem.AddPhoneNumber("123-456-7890");

            // Simulate sending notifications
            emailSystem.SendEmail(message);
            smsSystem.SendMessage(message);
        }
    }
```

```
  // Email notification system
public class EmailNotificationSystem
{
    private List<string> emailAddresses = new List<string>();

    public void Subscribe(string emailAddress)
    {
        emailAddresses.Add(emailAddress);
    }

    public void SendEmail(string message)
    {
        foreach (var email in emailAddresses)
        {
            Console.WriteLine($"Sending Email to {email}: {message}");
        }
    }
}

// SMS notification system
public class SMSNotificationSystem
{
    private List<string> phoneNumbers = new List<string>();

    public void AddPhoneNumber(string phoneNumber)
    {
        phoneNumbers.Add(phoneNumber);
    }

    public void SendMessage(string phoneNumber, string message)
    {
        foreach (var phone in phoneNumbers)
        {
            Console.WriteLine($"Sending SMS to {phone}: {message}");
        }
    }
}
```

EJERCICIO 4 (30')

Tenemos una situación en la que un sistema de facturación tiene diferentes tipos de conectores para procesar pagos. Uno muy viejo mediante *sockets* y se quiere incorporar un nuevo subsistema que puede trabajar con tecnología REST.

Explique y justifique qué patrón(es) resuelve(en) mejor el problema

a. Muestre un diagrama de clases con la solución
b. Realice los cambios en el código

```
class Program
{
    static void Main(string[] args)
    {
        var socketPaymentProcessor = new SocketPaymentProcessor();
        var restPaymentProcessor = new RestPaymentProcessor();

        // Test processing payments without a common interface
        var paymentAmount = 100.0;
        socketPaymentProcessor.ExecutePayment("user1", paymentAmount);
        restPaymentProcessor.MakePayment("user2", paymentAmount);
    }
}

// Legacy payment processor using sockets
public class SocketPaymentProcessor
{
    public void ExecutePayment(string username, double payment)
    {
        Console.WriteLine($"Executing payment for {username} through
legacy socket-based system: ${payment}");
    }
}

// Modern payment processor using REST
public class RestPaymentProcessor
{
    public void MakePayment(string username, double payment)
    {
        Console.WriteLine($"mmmm no quiero hacer esto…");
    }
}
```