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

EJERCICIO 1 (30')

Se tiene una aplicación para calcular el costo de envío de paquetes. Al principio, la aplicación tiene un método simple que toma en cuenta el peso del paquete y el tipo de envío (terrestre, aéreo, marítimo) para calcular el costo. A medida que la aplicación crece, y se agregan nuevos métodos de envío (uber, globo, PeYe…) se vuelve evidente que sería útil refactorizar este código para hacerlo más modular y fácil de mantener.

Analiza el siguiente código.

1- Identifica que antipatrón se está generando.
2- Refactorice el código.
   a. Explique y justifique qué patrón resuelve mejor el problema
   b. Muestre un diagrama de clases con la solución
   c. Realice los cambios en el código

```
public class ShippingCalculator
{
    public enum ShippingType
    {
        Ground,
        Air,
        Sea,
      ...,
      ...
    }

    public double CalculateShippingCost(double weight, ShippingType type)
    {
        double cost = 0;

        switch (type)
        {
            case ShippingType.Ground:
                cost = weight * 1.5;
                break;
            case ShippingType.Air:
                cost = weight * 2.5;
                break;
            case ShippingType.Sea:
                cost = weight * 1.2;
                break;
            case ...:
                cost = weight * ...;
                break;
            . . . .
        }

        return cost;
    }
}
```

EJERCICIO 2 (30')

UCU ha adquirido la sofisticada máquina de café Marley2.0. Esta máquina es conocida por su capacidad de preparar varios tipos de café y brindar a los usuarios la libertad de personalizar sus bebidas mediante diversas combinaciones.

Se nos ha encomendado la tarea de desarrollar un software que facilite la interacción de los usuarios con la máquina. La flexibilidad muy importante, ya que los usuarios deben poder elegir y combinar ingredientes a su gusto. Además, el software deberá calcular el costo de cada bebida basado en las opciones seleccionadas.

1- Refactorice el código.
2- Explique y justifique qué patrón resuelve mejor el problema
    a. Muestre un diagrama de clases con la solución
    b. Realice los cambios en el código

```
namespace uy.edu.ucu.andis1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Do you want coffee? (yes/no):");
            string coffee = Console.ReadLine();

            Console.WriteLine("Do you want milk? (yes/no):");
            string milk = Console.ReadLine();

            Console.WriteLine("Do you want sugar? (yes/no):");
            string sugar = Console.ReadLine();

            double cost = 0;

            if (coffee == "yes")
            {
                cost += 1.0;
            }

            if (milk == "yes")
            {
                cost += 0.5;
            }

            if (sugar == "yes")
            {
                cost += 0.2;
            }

            Console.WriteLine("Total cost: $" + cost);
        }
    }
}
```

EJERCICIO 3 (30')

Un programa de administración de una tienda en línea necesita un mecanismo para crear diferentes tipos de productos, como libros, ropa y electrónicos. Actualmente, el programa utiliza un enfoque directo para crear instancias de los productos, lo que hace que el código sea difícil de mantener y extender.

1- Refactorice el código.
2- Explique y justifique qué patrón resuelve mejor el problema
    a. Muestre un diagrama de clases con la solución
    b. Realice los cambios en el código

```
namespace uy.edu.ucu.andis1
{

      public class Libro
      {
          String nombre = "Libro";
          public string ObtenerNombre()
          {
              return nombre;
          }

          public decimal CalcularPrecio()
          {
              return 10.0;
          }
      }

      public class Ropa
      {
          String nombre = "Ropa";
          public string ObtenerNombre()
          {
              return nombre;
          }

          public decimal CalcularPrecio()
          {
              return 20.0;
          }
      }

      public class Electronico
      {
          String nombre = "Electrónico";
          public string ObtenerNombre()
          {
              return nombre;
          }

          public decimal CalcularPrecio()
          {
              return 30.0;
          }
      }
```

```
    class Program
    {
        static void Main()
        {
            Libro libro = new Libro();
            Ropa ropa = new Ropa();
            Electronico electronico = new Electronico();

            Console.WriteLine(libro.ObtenerNombre() + ": $" +
    libro.CalcularPrecio());
            Console.WriteLine(ropa.ObtenerNombre() + ": $" +
    ropa.CalcularPrecio());
            Console.WriteLine(electronico.ObtenerNombre() + ": $" +
    electronico.CalcularPrecio());
        }
    }
}
```