Ejercicios de Clases y objetos, polimorfismo y herencia y métodos

Los conceptos de clase y objeto, polimorfismo y herencia son la clave para comprender la programación orientada a objetos. Además también practicaremos con los métodos. Estos ejercicios nos ayudarán a asentar los conocimientos de la teoría.

Ejercicio 1

Siguiendo el ejemplo de teoría, vamos a crear un nuevo proyecto en Visual Studio llamado **AlmacenDeFiguras**. Vamos a crear varias clases:

1. Main: encargada de hacer pruebas con las clases geométricas.
2. Rectangulo: define un objeto rectángulo.
3. Circulo: define un objeto círculo.
4. Triángulo: define un objeto triángulo.

La aplicación creará tres figuras geométricas y el usuario podrá pedir información de cada una ellas y además comparar el área del rectángulo y el triángulo. Podemos acceder a las variables y mostrarlas por consola de la siguiente forma:

La respuesta al pedir información sobre las figuras tendrá este formato:

“Rectángulo: rojo”

“base: 2”

“altura: 4”

“area: 8”

La respuesta a la comparación debe ser una de estas dos:

“El rectángulo tiene un área mayor”

“El triángulo tiene un área mayor”

Para empezar, en nuestra clase Triangulo, crearemos 4 atributos, 3 de tipo **float** : **base, altura** y **área**, y 1 de tipo **string** : **color**.

A continuación crearemos dos constructores: uno sin que no reciba parámetros y otro que reciba dos parámetros, la **base** y la **altura**, en ellos inicializaremos los atributos.
Además deberemos crear un método que llamaremos **calcularArea()**, para calcular el área de un triángulo : base * altura /2.

Para el Círculo, crearemos 3 atributos, 2 de tipo **float**: **radio** y **área**, y uno de tipo **string** : **color**.

A continuación crearemos dos constructores uno que no reciba parámetros y otro que reciba el **radio** como parámetro, en ellos inicializaremos los atributos.
Además crearemos también el método **calcularArea()**, la fórmula para obtener el área de un círculo es : Math.PI * radio * radio.

Para el Rectángulo, crearemos también 4 atributos, 3 de tipo **float** : **base, altura** y **área**, y 1 de tipo **string** : **color**.
A continuación crearemos dos constructores: uno sin que no reciba parámetros y otro que reciba dos parámetros, la **base** y la **altura**, en ellos inicializaremos los atributos.
Además deberemos crear un método que llamaremos **calcularArea()**, para calcular el área de un triángulo : **base * altura**.

En la clase Principal debemos crear una instancia de cada objeto: **rectángulo, círculo** y **triángulo.**
Aquí dejamos el código perteneciente al método **main** de la clase en la que debemos probar el resto de clases.
Añadiremos también a esta clase un método que compare las áreas de un **triángulo** y un **rectángulo** y devolveremos cuál de los dos tiene mayor área.
Dentro de los **cases** debemos incluir el código necesario para consultar los atributos de las respectivas figuras para que tengan un formato semejante al que vimos al principio de este documento.

```
static void Main(string[] args)
       {
           int operacion;
           bool salir = false;
           // CREAR OBJETOS

           // RELLENAR

      while (!salir)
      {
       Console.WriteLine("Elige el tipo de figura que quieres consultar: ");
```

```
        Console.WriteLine("1 - Rectángulo / 2 - Círculo / 3 - Triángulo /" +
" 4 -   Comparar" + " 0 - Salir");
        operacion = Convert.ToInt32(Console.ReadLine());
        switch (operacion)
                {
                    case 1:
                        // RELLENAR
                        break;
                    case 2:
                        // RELLENAR
                        break;
                    case 3:
                        // RELLENAR
                        break;
                    case 4:
                        // RELLENAR
                        break;

                    case 0:
                        salir = true;
                        break;
                }
            }
```

Ejercicio 2

Ahora tenemos todas las clases desprotegidas sin modificadores. Vamos a añadir los modificadores **private** apropiados en las variables campo de las clases **Rectangulo**, **Circulo** y **Triangulo**. Por lo tanto, necesitaremos implementar los getters y setters para cada una de nuestras variables.

En la clase principal también tendremos que cambiar la forma de consultar las variables, ya que ahora son privadas.

Ejercicio 3

Una práctica muy común en los objetos es crear un método **toString()** que devuelve información sobre el objeto (valor de sus campos) en un String para ayudar a los desarrolladores a depurar el código. Mandan el resultado de **toString()** a la consola y pueden saber lo que contiene en cierto momento.

Actualmente casi todos los IDE's tienen herramientas de depuración o *Debug* que nos simplifica mucho la vida. Pero aun así usar el **toString()** puede ser muy útil.

Crear el método toString() en cada una de las clases de figuras geométricas. Sustituye la información que pasábamos a la consola por la que devuelve el método **toString()** en cada uno de los **case** de Principal.cs .

Tendrá que devolver la siguiente estructura:

"Rectángulo: rojo / base: 3 / altura: 2 / área: 6"

NOTA: Al trabajar con Strings podemos concatenar las cadenas usando el operador "+".

Ej: String cadena = "Hola" + persona.getNombre() + ", ¿qué tal estás?";

El resultado si persona.getNombre = "Ramón" sería: "Hola Ramón, ¿qué tal estás?"

Ampliación Ejercicio 3

Ahora deberemos realizar en nuestra clase principal las modificaciones necesarias para utilizar el nuevo método que nos permite obtener una cadena con las características de la figura.

Ejercicio 4

Vamos a modificar el método **compararRectanguloTriangulo**, incluiremos algunos cambios para modificar desde dentro del método los atributos de los respectivos objetos y comprobar los resultados, es decir, debemos comprobar si se propagan los cambios a los objetos que se pasan en los parámetros.

¿Qué ocurre con los cambios?

Ampliación Ejercicio 4

A continuación crearemos un nuevo método que compruebe si el área de un **rectángulo** está dentro de un límite. Los parámetros que debemos pasar son el **rectángulo** y el **límite.**

Ahora realizaremos los mismos cambios del ejercicio anterior, es decir, comprobaremos si haciendo cambios dentro de este método se propagan los cambios a la variable **límite**, es decir, la comprobación se realizará para la variable **límite** no para el objeto **rectángulo**.

¿Qué ocurre con los cambios?

Ejercicio 5

Crea un nuevo proyecto en el que la clase que se creará con este proyecto será nuestra clase Program. En ella crearemos dos métodos públicos en los que aplicaremos el **Polimorfismo Ad hoc** de **Sobrecarga**.

El primer método recibirá como parámetro un **entero** y devolverá un **array de entero** que contendrá los 15 primeros múltiplos de este número. Este método se llamará **múltiplos**.

```
public static int[] Multiplos(int p_numero)
{
    int[] p_multiplos = new int[15];
    for (int i = 1; i <= 15; i++)
    {
        p_multiplos[(i - 1)] = p_numero * i;
    }
    return p_multiplos;
}
```

El segundo método que crearemos será de tipo **bool** y recibirá dos parámetros que deben ser de tipo entero. Comprobaremos si el segundo número que pasamos como parámetro es múltiplo del primero.

```
public static bool Multiplos(int p_numero1, int p_numero2)
{
    if (p_numero2 % p_numero1 == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

En el método **main** le pediremos al usuario por consola que nos indique qué número quiere que se saquen sus primeros 15 múltiplos y llamaremos al método de los múltiplos para realizar la operación.

```
Console.WriteLine("Indica un número ");
int resp = Convert.ToInt32(Console.ReadLine());
string salida = "Los 15 primeros múltiplos del número " + resp + " son: " + "\n";
int[] v_multiplos = (int[])Multiplos(resp);
salida = salida + v_multiplos[0];
for (int i = 1; i < 15; i++)
{
    salida = salida + "-" + v_multiplos[i];
}
Console.WriteLine(salida + "\n");
```

A continuación procederemos a pedirle al usuario por consola qué números quiere comprobar si son múltiplos entre sí y realizaremos la llamada a otro método.

```
Console.WriteLine("Ahora vamos a comprobar si dos números son múltiplos");
Console.WriteLine("Indica un número ");
int num1 = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Indica otro número ");
int num2 = Convert.ToInt32(Console.ReadLine());
bool son_multiplos = Multiplos(num1, num2);
if (son_multiplos)
{
    Console.WriteLine("El número " + num2 + " es múltiplo del número " + num1 + "\n");
}
else
{
    Console.WriteLine("Los números " + num1 + " y " + num2 + " no son múltiplos" + "\n");
}
```

Ampliación Ejercicio 5

Vamos a crear una variable de tipo **string** y mediante el método **Parse** dentro de **Int32** le asignaremos su valor a una variable de tipo entero. Tras realizar esta conversión explícita, crearemos una variable de tipo double a la que le daremos el valor de la variable de tipo int creada anteriormente a partir del **string**.

```
Console.WriteLine("Indica un numero");
string v_variable = Console.ReadLine();
int v_num;
v_num = Int32.Parse(v_variable);
double v_double = v_num;
Console.WriteLine("Conversión explicita. Número:" + v_num + "\n");
Console.WriteLine("Conversión implicita. Número:" + v_double + "\n");
```

Ejercicio 6

Como vimos en teoría, el **polimorfismo universal de genericidad** nos permite utilizar los tipos genéricos, es decir, podemos crear clases sin especificar el tipo de datos. Así podremos utilizar esta clase para definir instancias de distintos tipos de la misma clase.

Para comenzar, crearemos una nueva clase que llamaremos **Contenedor** y será de tipo genérico y tendrá 3 atributos privados, uno de tipo **T**, genérico, que se llamará *contenido*, uno de tipo **string** llamado *nombre* y uno de tipo **int** llamado tamaño.

```
class Contenedor<T>
{
    private T contenido;
    private string nombre;
    private int tamaño;
```

En esta clase crearemos un constructor que reciba tres parámetros para instanciar los atributos de nuestra clase.

```
public Contenedor(T p_contenido, string p_nombre, int p_tamaño)
{
    contenido = p_contenido;
    nombre = p_nombre;
    tamaño = p_tamaño;
}
```

Además vamos a sobreescribir el método **ToString** dentro de la clase **Contenedor** que nos devuelva el **nombre** y el **tamaño** de nuestro contenedor.

```
public override string ToString()
{
    return "Nombre: " + this.nombre + " / Tamaño: " + this.tamaño;
}
```

Para terminar nuestro ejercicio nos dirigimos al método **Main** en nuestra clase **principal** y crearemos una instancia de nuestra clase contenedor de tipo **int**, llamando al constructor y pasándole los parámetros necesarios. Para terminar mostraremos por consola lo que nos devuelva el método **ToString**.

```
Contenedor<int> v_contenedor = new Contenedor<int>(5, "Mi Contenedor", 100);
Console.WriteLine(v_contenedor.ToString() + "\n");
```

Ampliación Ejercicio 6

Vamos a crear una instancia de nuestra clase contenedor para comprobar que mediante una clase genérica es posible crear instancias del tipo que queramos.

Debemos crear instancias de los siguientes tipos: **string**, **bool** y **double** y mostrar por pantalla lo que nos devuelva el método **ToString**. Es necesario cambiar los parámetros que le pasamos al constructor para poder distinguir las instancias de cada clase al mostrarlas por pantalla.

```
Contenedor<string> v_contenedor1 = new Contenedor<string>("Hola", "Mi Contenedor string", 100);
Console.WriteLine(v_contenedor1.ToString() + "\n");
Contenedor<bool> v_contenedor2 = new Contenedor<bool>(true, "Mi Contenedor bool", 200);
Console.WriteLine(v_contenedor2.ToString() + "\n");
Contenedor<double> v_contenedor3 = new Contenedor<double>(1.525, "Mi Contenedor double", 50);
Console.WriteLine(v_contenedor3.ToString() + "\n");
```

Ejercicio 7

En este ejercicio vamos a crear una estructura de clases en la que aplicaremos los conceptos de **herencia** que hemos visto en el tema.

Para empezar, crearemos una nueva clase **Coche** que será nuestra clase base, es decir, de esta clase van a heredar el resto de las clases de nuestra estructura. Dentro de esta clase vamos a crear 4 atributos, dos de tipo **string**: **matricula** y **marca** y otros dos de tipo **int**: **caballos** y **velocidad**. Los atributos deben ser de tipo protected.

```
class Coche
{
    protected string matricula;
    protected string marca;
    protected int caballos;
    protected int velocidad;
}
```

A continuación de la declaración de las variables crearemos un constructor para nuestra clase **Coche** que recibe tres parámetros para instanciar nuestros atributos.

```
public Coche(string p_matricula, string p_marca, int p_caballos)
{
    this.matricula = p_matricula;
    this.marca = p_marca;
    this.caballos = p_caballos;
}
```

Incluiremos también un método que llamaremos **Acelerar** de tipo **void** y lo marcaremos como virtual. En él comprobaremos si la velocidad es menor que 160, sumaremos 40 a la **velocidad** y mostraremos la velocidad que tiene nuestro coche por pantalla.

Nota: **Modificador del parámetro out.** La palabra clave `out` hace que los argumentos se pasen por referencia. Hace que el parámetro formal sea un alias para el argumento, que debe ser una variable. En otras palabras, cualquier operación en el parámetro se realiza en el argumento. Esto es como la palabra clave `ref`, salvo que `ref` requiere que se inicialice la variable antes de pasarla. Para usar un parámetro `out`, tanto la definición de método como el método de llamada deben utilizar explícitamente la palabra clave `out`.

```
public virtual void Acelerar(out int p_velocidad)
{
    if (this.velocidad < 160)
    {
        this.velocidad = this.velocidad + 40;
    }
    p_velocidad = this.velocidad;
}
```

Crearemos otro método de tipo **void** y que marcaremos como **virtual** en nuestra clase **coche**. A este método lo llamaremos **Frenar** y se encargará de reducer la **velocidad** de nuestro coche si ésta es mayor que **0**. Debemos tener en cuenta que al restar el valor de la **velocidad** no puede ser menor que **0**. Por lo tanto, deberemos utilizar las condiciones necesarias para solucionarlo. Al final de este método mostraremos también por pantalla la **velocidad** actual de nuestro **coche**.

```
public virtual void Frenar(int p_cantidad, out int p_velocidad)
{
    if ((this.velocidad > 0) && (this.velocidad - p_cantidad > 0))
    {
        this.velocidad = this.velocidad - p_cantidad;
    }
    else
    {
        this.velocidad = 0;
    }
    p_velocidad = this.velocidad;
}
```

La siguiente clase que vamos a crear va a ser **CocheElectrico**, que heredará de **Coche**. Añadiremos un parámetro de tipo **int** y **público** que se llamará **carga**. Además debemos crear un constructor que haga referencia al constructor de la clase **base**, pasándole todos los parámetros que recibe el constructor de la clase **Coche** y además también le pasaremos el atributo **carga**.

```
class CocheElectrico : Coche
{
    public int carga;

    2 referencias
    public CocheElectrico(int carga, string matricula,
    string marca, int caballos) : base(matricula, marca, caballos)
    {
        this.carga = carga;
    }
```

Lo siguiente que haremos en nuestra clase **CocheEléctrico** es sobreescribir el método **Acelerar** de la clase base de modo que sólo podamos acelerar si la **carga** de nuestro **CocheElectrico** es mayor que **0**. Además debemos restar **20** a la **carga** de nuestro **coche** cuando llamemos al método de la clase padre y debemos comprobar que al restar no se tome un valor negativo en el atributo **carga**, solucionando esto como hicimos anteriormente.

```
public override void Acelerar(out int p_velocidad)
{
    if (this.carga > 0)
    {
        base.Acelerar(out p_velocidad);
        if (this.carga - 20 > 0)
        {
            this.carga = this.carga - 20;
        }
        else
        {
            this.carga = 0;
        }
    }
    else
    {
        p_velocidad = this.velocidad;
    }
}
```

También sobreescribiremos el método **Frenar**, lo que nos permitirá cargar nuestro **CocheElectrico**, es decir, si nuestra carga es menor que **100** y frenamos, se cargará nuestro coche **10** unidades. También debemos tener en cuenta que la carga del coche no puede superar **100**.

```
public override void Frenar(int p_cantidad, out int p_velocidad)
{
    if (this.velocidad > 0)
    {
        int velocidad = this.velocidad;
        base.Frenar(40, out p_velocidad);
        if ((p_velocidad != velocidad) && (this.carga + 10 <= 100))
        {
            this.carga = this.carga + 10;
        }
    }
    else { p_velocidad = this.velocidad; }
}
```

A continuación, en nuestra clase **Principal** crearemos un **CocheElectrico** y aceleraremos 2 veces, mostrando cada vez la carga de nuestro coche. Además para acabar utilizaremos el método frenar y volveremos a mostrar la **carga** de nuestro coche.

```
CocheElectrico coche1 = new CocheElectrico(50, "123456", "Volvo", 500);
coche1.Acelerar(out int velocidad);
Console.WriteLine("Aceleramos a velocidad de " + velocidad
+ " y la carga actual es " + coche1.carga + "\n");
coche1.Acelerar(out int velocidad1);
Console.WriteLine("Aceleramos de nuevo, a velocidad de " + velocidad1
+ " y la carga actual es " + coche1.carga + "\n");
Console.WriteLine("Indica la cantidad que quieres frenar");
int p_cantidad = Convert.ToInt32(Console.ReadLine());
coche1.Frenar(p_cantidad, out int velocidad2);
Console.WriteLine("Frenamos a " + velocidad2 + " y la carga actual es " + coche1.carga + "\n");
```

En este ejercicio aplicaremos la herencia múltiple. Crearemos una nueva interfaz llamada **IVehiculoPropio**, que implementará nuestra clase **CocheElectrico**. Esta interfaz tendrá dos métodos **MostrarDueño** y **AñosPertenencia**. Estos métodos requerirán añadir dos atributos a nuestra clase **Coche** y **CocheElectrico**, uno de tipo **string dueño** y otro de tipo **int añoAdquisición**, ambos será privados (debemos instanciar estos atributos en el constructor de nuestra clase, es decir, habrá que añadir parámetros).

```
public interface IVehiculoPropio
{
    0 referencias
    void MostrarDueño();
    2 referencias
    void AñosPertenencia();
}
```

El código de los constructores:

```
public Coche(string p_matricula, string p_marca, int p_caballos, string p_dueño, int p_año)
{
    this.matricula = p_matricula;
    this.marca = p_marca;
    this.caballos = p_caballos;
    this.dueño = p_dueño;
    this.añoAdquisicion = p_año;
}
```

```
public CocheElectrico(int carga, string matricula,string marca,
    int caballos, string dueño, int año) :
    base(matricula, marca, caballos, dueño, año)
{
    this.carga = carga;
}
```

Debemos implementar en nuestra clase **Coche** y **CocheElectrico** todos los métodos de la interfaz. Estos métodos deben mostrar por pantalla el **dueño** y los **años** que pertenece a cierto **dueño** el **coche**. También deberemos mostrar el resultado de restar el **año actual** y la **fecha de compra del coche**.

```
class CocheElectrico : Coche, IVehiculoPropio

        public string GetDueño()
        {
            return this.dueño;
        }

         public int GetAñoAdquisicion()
         {
             return this.añoAdquisicion;
         }

public void MostrarDueño()
{
    Console.WriteLine("El dueño de este coche es " + this.GetDueño() + "\n");
}


public void AñosPertenencia()
{
    DateTime fecha_hoy = DateTime.Now;
    int año = fecha_hoy.Year;

    Console.WriteLine("Este coche pertenece " + (año - this.GetAñoAdquisicion()) +
        " años a su dueño " + this.GetDueño() + "\n");
}
```

Para acabar nuestros ejercicios, comprobaremos que funcionan correctamente los métodos que hemos implementado en la clase **CocheElectrico**. Los probaremos en la clase **Principal**, mediante las llamadas correspondientes, deberemos crear instancias de nuestras clases.

```
CocheElectrico coche2 = new CocheElectrico(100, "COCHE2", "Audi", 200, "Ana", 2020);
coche2.MostrarDueño();
coche2.AñosPertenencia();
```

Ejercicio 8

Aunque ya estamos familiarizados con los métodos, en este tema hemos profundizado más en ellos y hemos descubierto nuevas formas de utilizarlos.

Para empezar comenzaremos creando un nuevo proyecto que llamaremos **Ejercicio8**. La clase que se crea cuando creamos nuestro proyecto será nuestra clase **Program**.

Nuestro programa mostrará un menú con las siguientes opciones:

1.- Números pares

2.- Eliminar vocales

3.- Comida

0.- Salir

Mientras que el usuario no elija la opción "0" el programa continuará ejecutándose.

Apartado 1. Números pares

Dentro de nuestra clase Program vamos a crear dos métodos, para aplicar el concepto de **Lanzadera**, es decir, un método sin parámetros que llame al otro método con unos parámetros por defecto, el segundo método recibirá parámetros cuando lo llamemos.

El método que vamos a crear se encargará de calcular números pares, es decir, un método que devolverá un **array** de **int** que contendrá los números pares hasta un número **n** que debemos pasarle en los parámetros. Dentro de este método debemos recorrer los números hasta el número **n**, comprobar uno a uno si son pares y, si lo son, añadirlos a nuestro **array**. Finalmente, tendremos que devolver nuestro **array**.

```
static int[] NumerosPares(int n)
{
    int[] resultado = new int[n];
    int i = -1;
    for (int j = 0; j <= n; j++)
    {
        if ((j % 2) == 0)
        {
            i++;
            resultado[i] = j;
        }
    }
    return resultado;
}
```

El método lanzadera se encargará de llamar a este método para calcular por defecto los números pares hasta **100.**

```
static int[] NumerosPares()
{
    return NumerosPares(100);
}
```

Ahora en el método **Main** crearemos un array de tipo int que se llamará **pares** y una variable de tipo string que se llamará **salida** y crearemos un bucle para mostrar por pantalla los números pares hasta el 100.

```
int[] pares = NumerosPares();
string salida;
Console.WriteLine("La relación de números pares hasta 100 es: " + "\n");
salida = "(" + pares[0];
for (int i = 1; pares[i] < 100; i++)
{
    salida = salida + "," + pares[i];
}
salida = salida + ")";
Console.WriteLine(salida + "\n" + "\n");
```

Posteriormente preguntaremos al usuario por consola hasta qué número quiere que se busquen los números pares.

```
Console.WriteLine("Indica hasta qué número quieres obtener números pares ");
int resp = Convert.ToInt32(Console.ReadLine());
pares = NumerosPares(resp);
Console.WriteLine("La relación de números pares hasta {0} es: " + "\n", resp);
salida = "(" + pares[0];
for (int i = 1; i < pares.Length; i++)
{
    salida = salida + "," + pares[i];
}
salida = salida + ")";
Console.WriteLine(salida + "\n" + "\n");
```

Apartado 2. Eliminar vocales

En este ejercicio vamos a poner en práctica lo aprendido acerca de pasar parámetros por referencia.

Crearemos un nuevo método de tipo **void** que se llamará **eliminarVocales** y recibirá una **referencia a un string**.

Dentro del método crearemos dos strings **cadena** y **sinvocales**, donde **cadena** tomará por parámetro el string de referencia del método y convertirá la palabra con la primera letra a mayúscula, mientras que **sinvocales** será un string vacío.

```
static void eliminarVocales(ref string palabra)
{
    string cadena = palabra.ToUpper();
    string sinvocales = "";
```

A continuación crearemos un bucle para recorrer las posiciones del string que tiene por parámetro el método desde **0** hasta la longitud del mismo. Mediante el método **SubString(índice, longitud)** comprobaremos que dicha subcadena no se trate de una vocal, entonces lo añadiremos a otro **string**. Al acabar el bucle igualaremos la referencia al **string** a la que solo contiene consonantes.

```
for (int i = 0; i < palabra.Length; i++)
{
    switch (cadena.Substring(i, 1))
    {
        case "A":
            break;
        case "E":
            break;
        case "I":
            break;
        case "O":
            break;
        case "U":
            break;
        default:
            sinvocales = sinvocales + palabra.Substring(i, 1);
            break;
    }
}
```

En nuestro método **Main** vamos a crear una instancia de nuestra clase y una variable de tipo **string** que le pasaremos como referencia al método **eliminarVocales**. Para comprobar los cambios, mostraremos la cadena por pantalla antes y después de utilizar el método.

```
Console.WriteLine("Indica una palabra a la que eliminar las vocales ");
string palabra = Console.ReadLine();
eliminarVocales(ref palabra);
Console.WriteLine("El resultado es " + palabra + "\n");
```

Apartado 3. Comida

En este ejercicio veremos cómo usar las funciones asíncronas y el uso de la librería Task.

Comenzaremos por importar las librerías necesarias para el ejercicio: *System.Threading* y *System.Threading.Tasks*.

```
using System.Threading;
using System.Threading.Tasks;
```

En este ejercicio crearemos dos métodos asíncronos, el primer método se llamará **OfrecerComidaAsync**. Será un método **async** de tipo **Task**, que creará una variable de tipo **string** que llamaremos **result**, a la que se le asigne la espera mediante **await** a una llamada al método **BuscarComidaAsync**, que es el segundo método que crearemos. **OfrecerComidaAsync** también mostrará por pantalla la variable **result** y añadiremos la cadena "¿Te apetece comer algo?"

```
private async Task OfrecerComidaAsync()
{
    string result = await BuscarComidaAsync();
    Console.WriteLine(result);
    Console.WriteLine("¿Te apetece comer algo?");
}
```

BuscarComidaAsync deberá esperar mediante **await** y **Task.Delay** el tiempo que nosotros deseemos que se retrase la tarea en efectuarse y devolverá la siguiente cadena "He traído esto de la nevera".

```
private async Task<string> BuscarComidaAsync()
{
    await Task.Delay(2000);
    return "He traído esto de la nevera.";
}
```

Desde el método **Main**, de la clase **principal** crearemos una instancia de nuestra clase y llamaremos al método **OfrecerComidaAsync**. Como este es un método **async** debemos esperar a que termine o acabara antes el hilo principal que el resto, por lo que no se mostrará el resultado de realizar la llamada al método. Por tanto, debemos utilizar **Thread.Sleep**, para esperar al menos el mismo tiempo que esperamos dentro del método **BuscarComidaAsync**. Para comprobar que hemos terminado la ejecución, mostraremos por pantalla una cadena semejante a "Fin", para poder saber cuándo ha acabado de ejecutarse el código y si las llamadas se realizan correctamente. Si se muestra por pantalla **Fin** antes que las otras cadenas, nuestro código no funciona correctamente.

```
Program mi_clase = new Program();
mi_clase.OfrecerComidaAsync();
Thread.Sleep(2100);
Console.WriteLine("Fin" + "\n");
```

La salida por pantalla es la siguiente:

```
Elige opción.
1.- Números Pares
2.- Eliminar vocales
3.- Comida
0.- Salir

1
La relación de números pares hasta 100 es:

(0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,
78,80,82,84,86,88,90,92,94,96,98)


Indica hasta qué número quieres obtener números pares
10
La relación de números pares hasta 10 es:

(0,2,4,6,8)


Elige opción.
1.- Números Pares
2.- Eliminar vocales
3.- Comida
0.- Salir

2
Indica una palabra a la que eliminar las vocales
Luisa
El resultado es Ls

Elige opción.
1.- Números Pares
2.- Eliminar vocales
3.- Comida
0.- Salir

3
He traído esto de la nevera.
¿Te apetece comer algo?
Fin
```