Indique cuáles de las siguientes afirmaciones son verdaderas: 1 punto

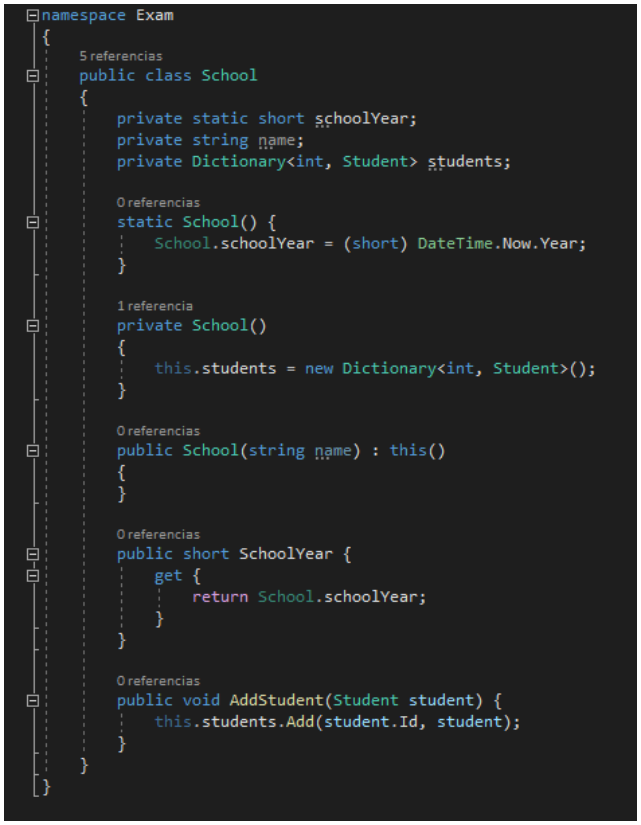
Teniendo en cuenta que el código se encuentra dentro del archivo Codigo.cs

```
namespace Exam
{
    5 referencias
    public class School
    {
        private static short schoolYear;
        private string name;
        private Dictionary<int, Student> students;

        0 referencias
        static School() {
            School.schoolYear = (short) DateTime.Now.Year;
        }

        1 referencia
        private School()
        {
            this.students = new Dictionary<int, Student>();
        }

        0 referencias
        public School(string name) : this()
        {
        }

        0 referencias
        public short SchoolYear {
            get {
                return School.schoolYear;
            }
        }

        0 referencias
        public void AddStudent(Student student) {
            this.students.Add(student.Id, student);
        }
    }
}
```

- [ ] El código en la imagen se compilará a lenguaje intermedio en tiempo de ejecución.
- [ ] Visual Studio se encargará de administrar la memoria utilizada por nuestro programa. Gracias a esto, no se requerirá de nuestra intervención para asignar y liberar memoria.
- [ ] El atributo "students" es de un tipo de dato escalar.
- [x] La clave / key del atributo "students" es de un tipo de dato escalar.
- [ ] Ninguna respuesta.

Indique cuáles de las siguientes afirmaciones son verdaderas: 1 punto

Responda en base al código en la imagen.



```
namespace Exam
{
    5 referencias
    public class School
    {
        private static short schoolYear;
        private string name;
        private Dictionary<int, Student> students;

        0 referencias
        static School() {
            School.schoolYear = (short) DateTime.Now.Year;
        }

        1 referencia
        private School()
        {
            this.students = new Dictionary<int, Student>();
        }

        0 referencias
        public School(string name) : this()
        {
        }

        0 referencias
        public short SchoolYear {
            get {
                return School.schoolYear;
            }
        }

        0 referencias
        public void AddStudent(Student student) {
            this.students.Add(student.Id, student);
        }
    }
}
```

- [ ] "School" es un bloque de memoria que se ha asignado y configurado para almacenar un conjunto de datos.
- [ ] "Exam" representa un conjunto de objetos agrupados lógicamente.
- [ ] La propiedad "SchoolYear" dará un error por intentar acceder a un atributo estático desde una propiedad de instancia.
- [ ] Si instancio un objeto de tipo "School" con el constructor ("static School()"), nunca se instanciará el diccionario, dando NullReferenceException al invocar al método AddStudent().
- [x] Ninguna respuesta.

Indique cuáles de las siguientes sobrecargas del método "AddStudent" son válidas: 1 punto

Responda en base al código en la imagen.

```
0 referencias
public void AddStudent(Student student) {
    this.students.Add(student.Id, student);
}
```

- [ ] public void AddStudent(Student s)
- [ ] public static void AddStudent(Student student)
- [x] public void AddStudent(out Student student)
- [x] public void AddStudent(Student student1, Stundent stundent2)
- [ ] Ninguna respuesta.

Marque cuál/es de las siguientes afirmaciones es verdadera: 1 punto

Responda en base al código en la imagen. Tenga en cuenta que DateTime.Now.Year devuelve un número de tipo "int".

```
public class School
{
    private static short schoolYear;
    private string name;
    private Dictionary<int, Student> students;

    static School() {
        School.schoolYear = (short) DateTime.Now.Year;
    }

    private School()
    {
        this.students = new Dictionary<int, Student>();
    }

    public School(string name) : this()
    {
        this.name = name;
    }

    public short SchoolYear {
        get {
            return School.schoolYear;
        }
    }

    public static bool operator == (School school, Student student)
    {
        foreach (Student item in this.students)
        {
            if (item.Id == student.Id)
            {
                return true;
            }
        }
        return false;
    }

    public static bool operator != (School school, Student student)
    {
        return !(School == student);
    }
}
```

- [ ] Usar el operador de casteo "(short)" es redundante ya que, al no haber pérdida de información, existe una conversión implícita.
- [ ] "Now" es un miembro no-estático de "DateTime".
- [x] La propiedad "SchoolYear" de todos los objetos de tipos "School" que instancie devolverán el mismo valor.
- [ ] La declaración de la sobrecarga del operador de igualdad dentro de la clase "School" es válida (no tiene errores).
- [ ] Ninguna respuesta.

Indique cuál/es de las siguientes afirmaciones sobre Windows Forms son verdaderas: 1 punto

- [x] Cuando se lanza el evento Load del formulario, los controles ya fueron instanciados pero aún no se muestran en pantalla.
- [x] La clase en el archivo ".Designer.cs" tiene un constructor, pero está en otro archivo.
- [ ] La clase en el archivo ".Designer.cs" no tiene constructor.
- [ ] El método ShowDialog() visualiza un formulario modal. Eso significa que estará contenido dentro de los límites de la ventana del formulario padre.
- [ ] Ninguna respuesta.

De acuerdo a lo visto en clase. Indique cuáles de las siguientes propiedades corresponde a cada tipo de colección: 1 punto

	Indexada por número de posición del elemento.	Indexada por clave / key.	Compuesta por pares clave-valor.	Todos sus elementos son de un tipo específico.	Puede estar compuesta por elementos de distintos tipos.
Stack	☐	☐	☐	☑	☑
ArrayList	☑	☐	☐	☐	☑
List	☑	☐	☐	☑	☐
SortedList	☑	☑	☑	☑	☐

Podemos utilizar la palabra "this" dentro de una clase para: 1 punto

- ☐ Hacer referencia a un atributo estático.
- ☐ Invocar a un constructor de la misma clase desde un método.
- ☐ Es una palabra reservada que se utilizará dentro del set de un indexador para indicar el valor recibido por el mismo.
- ☐ Consultar un atributo de la instancia dentro de la implementación de una conversión explícita.
- ☑ Ninguna respuesta.

Enumere y corrija (si los hay) el/los errores en el siguiente método: 1 punto

Describa cada error y cómo lo arreglaría. El operador está declarado dentro de la clase Producto, la cual contiene un atributo precio.

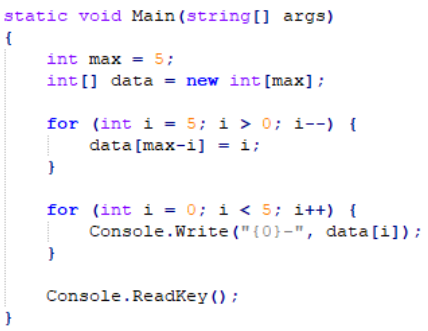
```
public static explicit operator + (Producto producto, double aumentoPrecio)
{
    this.precio += precio;

    return producto;
}
```

Es estático va producto. no this.
precio en realidad es aumentoPrecio.
Es un operador, no va explicit, va Producto en el retorno.

Marque cuál/es de las siguientes afirmaciones es verdadera con respecto al siguiente código: "Planeta unPlaneta = new PlanetaGaseoso( );" 1 punto

- ☑ Es válido si PlanetaGaseoso hereda de Planeta.
- ☐ Se perderán todos los datos almacenados en los atributos de la instancia de PlanetaGaseoso.
- ☐ Es válido si existe una sobrecarga del operador explicit en Planeta que recibe un PlanetaGaseoso y retorna un Planeta.
- ☐ Esto es el pilar de la programación orientada a objetos conocido como abstracción.

¿Qué muestra la salida del código en la imagen? 1 punto



```
static void Main(string[] args)
{
    int max = 5;
    int[] data = new int[max];

    for (int i = 5; i > 0; i--) {
        data[max-i] = i;
    }

    for (int i = 0; i < 5; i++) {
        Console.Write("{0}-", data[i]);
    }

    Console.ReadKey();
}
```

5-4-3-2-1-

Observaciones

Indique enunciado y respuesta sobre la que hará la observación, de otra forma el comentario no será tenido en cuenta.

Las preguntas no están numeradas, copie y pegue el texto.

Ingrese sus comentarios:

Preguntas de promoción

Los siguientes puntos se tendrán en cuenta en la corrección para alcanzar la promoción directa.

Lea atentamente los enunciados y las respuestas antes de responder.

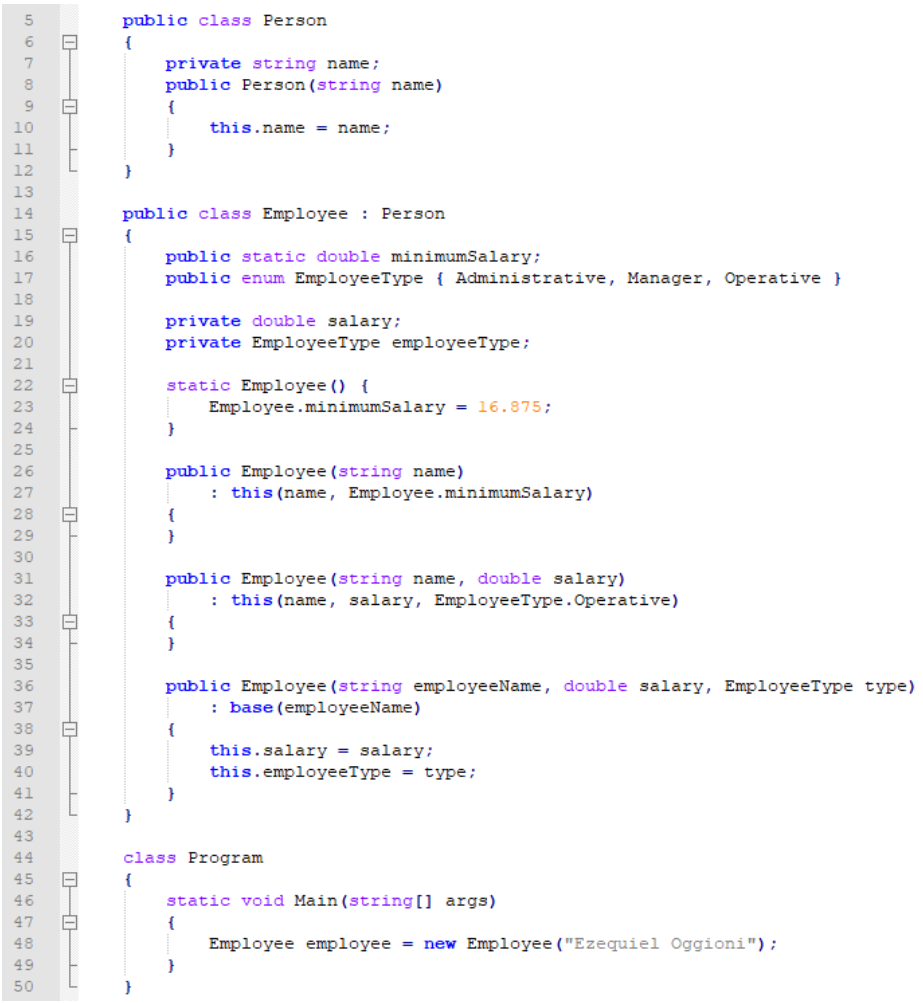
Relacione el/los modificadores que correspondan con la función que se describe: 1 punto

La respuesta puede ser la combinación de varios modificadores.

	public	abstract	sealed	protected	virtual	static	ninguno
Es un método cuya implementación puede ser invalidada por clases derivadas y puede accederse desde clases no-derivadas.	☑	☐	☐	☐	☑	☐	☐
Es una clase que no puede ser base, pero puede instanciarse.	☐	☐	☑	☐	☐	☐	☐
Es una clase que no puede ser derivada ni instanciarse.	☐	☐	☐	☐	☐	☑	☐
Es un método que no puede acceder a atributos de instancia y su implementación debe ser invalidada por la clase derivada obligatoriamente.	☐	☐	☐	☐	☐	☑	☐

Indique, en tiempo de ejecución, el orden en que se ejecutarán las líneas de código indicadas. 2 puntos

Responda en base al código en la imagen. Cuenta con el número de la línea de código a la izquierda de la imagen. El código NO tiene errores. Considere que los bloques vacíos ({ }) se ejecutan igual, como si estuviera debuggeando el código.



```
5   public class Person
6   {
7       private string name;
8       public Person(string name)
9       {
10          this.name = name;
11      }
12  }
13
14  public class Employee : Person
15  {
16      public static double minimumSalary;
17      public enum EmployeeType { Administrative, Manager, Operative }
18
19      private double salary;
20      private EmployeeType employeeType;
21
22      static Employee() {
23          Employee.minimumSalary = 16.875;
24      }
25
26      public Employee(string name)
27          : this(name, Employee.minimumSalary)
28      {
29      }
30
31      public Employee(string name, double salary)
32          : this(name, salary, EmployeeType.Operative)
33      {
34      }
35
36      public Employee(string employeeName, double salary, EmployeeType type)
37          : base(employeeName)
38      {
39          this.salary = salary;
40          this.employeeType = type;
41      }
42  }
43
44  class Program
45  {
46      static void Main(string[] args)
47      {
48          Employee employee = new Employee("Ezequiel Oggioni");
49      }
50  }
```

	Primera	Segunda	Tercera	Cuarta	Quinta	No se ejecuta
Línea #10	○	○	◉	○	○	○
Línea #39	○	○	○	◉	○	○
Línea #27	◉	○	○	○	○	○
Línea #28	○	○	○	○	◉	○
Línea #32	○	◉	○	○	○	○

Asociar cada descripción con el pilar de la programación orientada a objetos que corresponda. 1 punto

	Polimorfismo.	Encapsulamiento.	Abstracción.	Herencia.	Ninguna.
Permite crear clases más especializadas a partir de otras más generales.	○	○	○	◉	○
Consiste en ocultar los detalles de la implementación y proteger el acceso a datos.	○	◉	○	○	○
Consiste en seleccionar las características relevantes/importantes y comportamiento/operaciones en común dentro de un conjunto de objetos, definiendo nuevos tipos de entidades.	○	○	◉	○	○
Es la propiedad que tienen los objetos de permitir invocar genéricamente un comportamiento (método) cuya implementación será delegada al objeto correspondiente recién en tiempo de ejecución.	◉	○	○	○	○

En base al código en la imagen. Marque cuál/es de las siguientes afirmaciones son verdaderas. 1 punto

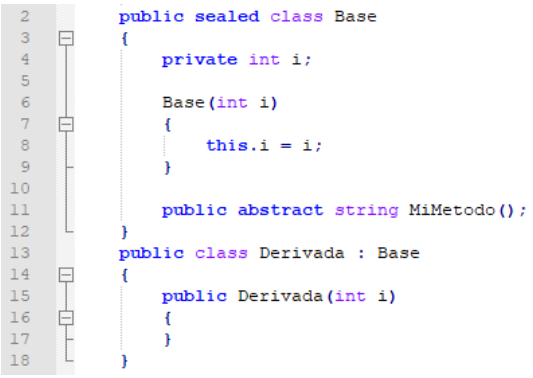
IMPORTANTE: Tenga en cuenta sólo el caso mencionado en la respuesta, como si los otros dos casos no existieran en el código o estuvieran comentados.

```
class Resultado
{
    private int valor;

    public Resultado(int valor)
    {
        this.valor = valor;
    }

    public int Valor
    {
        get
        {
            return this.valor;
        }
        set
        {
            this.valor = value;
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        Resultado objetoResultado = new Resultado(0);
        int primerOperando = 3;
        int segundoOperando = 4;

        // Caso 1
        int resultado = objetoResultado.Valor;
        SumarUno(out resultado, primerOperando, segundoOperando);
        objetoResultado.Valor = resultado;

        // Caso 2
        SumarDos(objetoResultado.Valor, primerOperando, segundoOperando);

        // Caso 3
        SumarTres(objetoResultado, primerOperando, segundoOperando);

        Console.WriteLine(objetoResultado.Valor);
        Console.ReadKey();
    }

    static void SumarUno(out int resultado, int primerOperando, int segundoOperando)
    {
        resultado = primerOperando + segundoOperando;
    }

    static void SumarDos(int resultado, int primerOperando, int segundoOperando)
    {
        resultado = primerOperando + segundoOperando;
    }

    static void SumarTres(Resultado resultado, int primerOperando, int segundoOperando)
    {
        resultado.Valor = primerOperando + segundoOperando;
    }
}
```

- [x] objetoResultado es un reference type.
- [x] En el Caso 1 la salida por pantalla será "7".
- [ ] En el Caso 2 la salida por pantalla será "null".
- [ ] En el Caso 3 la salida por pantalla será "0".
- [ ] Ninguna respuesta.

Identificar, enumerar, justificar e indicar cómo corregiría el/los errores presentes en el código de la imagen. 2 puntos

Indique el número de la línea de código donde se encuentra cada error.



```
2   public sealed class Base
3   {
4       private int i;
5
6       Base(int i)
7       {
8           this.i = i;
9       }
10
11      public abstract string MiMetodo();
12  }
13  public class Derivada : Base
14  {
15      public Derivada(int i)
16      {
17      }
18  }
```

Sacar el sealed poner abstract.
Agregar :base (i)
Sobrescribir el método o sacarle el abstract.
Hacer protected o public el constructor de la clase base.

Indicar el PRIMER mensaje mostrado por consola: 2 puntos

```
    class Persona
    {
        public virtual string Pensar()
        {
            return "ay... 20 dólares... yo quería maní...";
        }
    }
    class Alumno : Persona
    {
        public override string Pensar()
        {
            return "Bueno cerebro. Yo no te agrado y tú no me agradas, " +
                "pero vamos a hacer esto juntos.";
        }
    }
    class Profesor : Persona
    {
        public override string Pensar()
        {
            return "6 horas, 1 minuto, ascensión derecha, 14 grados, declinación 22
minutos... no hay aprobados...";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Stack<Persona> personas = new Stack<Persona>();
            personas.Push(new Profesor());
            personas.Push(new Persona());
            personas.Push(new Alumno());

            foreach (Persona persona in personas)
            {
                Console.WriteLine(persona.Pensar());
            }

            Console.ReadKey();
        }
    }
```

Bueno cerebro. Yo no te agrado y tú no me agradas, pero vamos a hacer esto juntos.

Confirmación de entrega

¡Llegaste hasta el final del examen!