Herencia: El Principio de Liskov define la manera en la que se debe utilizar la herencia en la programación orientada a objetos

++ O - - el valor de retorno debe ser del tipo que lo contiene

Clases estáticas:

- Las clases estática no participan en la herencia es decir una clase estática no puede ser base ni derivada

- Una clase estática solo debe contener métodos y atributos estáticos

- Las clases estáticas no pueden ser selladas ya que Implícitamente ya son selladas

- No se pueden instanciar

Clase selladas:

- Se pueden instanciar
- Pueden ser hijas (derivadas) pero no base

Clases abstractas:

- No pueden instanciarse
- Sirven para ser la base
- También pueden ser derivadas es decir hijas

Queue y Stack:

- no se pueden indexar

- Las colecciones de tipo Queue se procesan con orden FIFO (Primero en Entrar, Primero en Salir - first in first out -).

- Las colecciones de tipo Stack se procesan con orden LIFO (Último en Entrar, Primero en Salir - Last in, First out -).

- Se guardan en el heap, lo que se guarda en el stack es la dirección de memoria donde vive la colección en el heap

Garbage collector:

- El garbage collector libera la memoria almacenada en el segmento Heap, no en el Stack. El segmento Stack es liberado automáticamente.

Indexadores:

- No es un requisito que la clase tenga una colección o array como atributo para poder ser indexada. De nuevo, depende de la necesidad y del criterio de indexación diseñado.

- Pueden recibir más de un parámetro, es decir, pueden trabajar con más de un índice

- Se pueden sobrecargar

- Se puede indexar con valores que no sean numéricos

- Los indexadores no pueden ser estáticos

Formularios:

- Son objetos

- Utilizan clases parciales, reparando una misma clase en distintos archivos

- El formulario dibujara de acuerdo a los valores que tenga cargados en sus atributos

- Un formulario puede heredar de otro formulario

- Las acciones sobre la interfaz de usuario se informan a través de eventos

- Sus constructores pueden tener parámetros

- Heredan de System.Windows.Forms.

- Poseen un constructor por defecto que inicializa todo lo que compone al form.

Common Language Runtime:
Un entorno de ejecución (runtime) es un programa encargado de administrar la ejecución de un programa. Permite abstraer a nuestras aplicaciones de la máquina sobre la que se ejecutan. Todos los programas creados con .NET se ejecutan sobre un runtime.

El runtime de .NET Framework y .NET5+ se conoce como Common Language Runtime(CLR)

Entre sus tareas se encuentra:

- Administrar el uso , asignación y liberación de memoria.
- Genera y compila código para que el programa pueda ejecutarse sobre una máquina en concreto.

- Sirve como capa de abstracción del hardware, permitiendo a los desarrolladores programar de forma agnóstica a la plataforma de destino.
- Manejo de errores en tiempo de ejecución.
- Provee funcionalidades para lenguajes orientados a objetos.
- Soporte para programación multi-hilo.
- Cuestiones de seguridad y rendimiento.
- Es el encargado de llamar al constructor estático

Namespaces:

- Su principal función es la de de organizar el código para reducir los conflictos entre nombres
- Es una agrupación lógica de tipos de datos
- Organización del código fuente

Los espacios de nombres(namespaces) son una agrupación lógica de clases y otros elementos del código fuente. así como clasificamos nuestros archivos dentro de distintos directorios del sistema operativo, podemos organizar el código fuente de c# y sus componentes en distintos espacios de nombres.
Su función principal es la de organización del código, permitiendo la reducción de conflictos por nombres duplicados y la posibilidad de trabajar en un mismo programa con componentes de distinta procedencia.

Directiva using:
La directiva using permite la especificación de una llamada a un método sin el uso obligatorio de un nombre completamente cualificado(nombre completo incluyendo el espacio de nombres)

Objeto:

- Se crean en tiempo de ejecución

Un objeto en programación representa algo de la vida real, como puede ser un producto o una cuenta bancaria, pero también puede ser algo más abstracto.

Abstracción:
La abstracción es la habilidad de abordar un concepto mientras se ignoran de forma segura algunos de sus detalles. Dependiendo del contexto y de la necesidad podemos trabajar a distintos niveles de abstracción, enfocándonos en distintos aspectos y trabajando con un mayor o menor nivel de detalle.

common type system:

- Define un conjunto común de "tipos" de datos orientados a objetos.

Constructores:

- Inicializar el estado de un objeto

Sobrecarga de métodos:

- Sobrecargar un método es para agregar funcionalidad

Si yo quiero sobrecargar este método:

```
0 referencias
public int calcularEdad(int diaNacimiento, int mesNacimineto, int anioNacimiento)
{
    return 1;
}
```

Si o si debo cambiar orden o cantidad o tipo de parámetros, es decir, si yo dejo todos los parámetros iguales y lo único que modifico es el valor de retorno o el modificador de acceso el método no se sobrecarga ejemplo:

```
0 referencias
public int calcularEdad(int diaNacimiento, int mesNacimineto, int anioNacimiento)
{
    return 1;
}
//Me tira error xq no lo puedo sobrecargar ya que no modifique nada de los parametros
//Lo unico que modifique fue su tipo de retorno pero eso no influye en la sobrecarga
0 referencias
public float calcularEdad(int diaNacimiento, int mesNacimineto, int anioNacimiento)
{
    return 1;
}
```

float ClaseDePrueba.calcularEdad(int diaNacimiento, int mesNacimineto, int anioNacimiento)

CS0111: El tipo 'ClaseDePrueba' ya define un miembro denominado 'calcularEdad' con los mismos tipos de parámetro

Mostrar posibles correcciones (Alt+Entrar o Ctrl+.)

```
//Aca lo mismo, no se sobrecarga xq no cambie nada en los parametros, solo cambie
//su modificador de acceso
0 referencias
private int calcularEdad(int diaNacimiento, int mesNacimineto, int anioNacimiento)
{
    return 1;
}
```

El compilador distingue los métodos que están sobrecargados comparando la lista de parámetros, tipo, número u orden de parámetros de entrada
Partial: El modificador partial permite separar la declaración de una misma clase en dos o más archivos.

Modificadores de acceso:

- Public: El acceso no está restringido. Se puede acceder a una clase o miembro public desde cualquier otro código en el mismo proyecto y otro proyecto que haga referencia a él.

- Private: El acceso está limitado al tipo contenedor. Solo se puede acceder a una clase o miembro privado desde la misma clase o struc.

- Protected: El acceso está limitado a la clase contenedora o a las derivadas de la clase contenedora. Se puede acceder al miembro protected desde la misma clase o su derivada, es decir, se usa para extender la visibilidad de una clase base a una derivada sin comprometer el encapsulamiento

- Internal: El acceso está limitado al proyecto actual, es decir, puedo acceder desde cualquier clase pero solo dentro un proyecto/ensamblado. Se puede acceder a una clase o miembro private desde cualquier código del mismo proyecto, pero no desde otro proyecto.

- Protected internal: El acceso está limitado al proyecto actual o las derivadas de la clase contenedora. Se puede acceder a una clase o miembro protected internal desde cualquier código en el proyecto en el que está declarado, o desde una clase derivada en otro proyecto.

- Private protected: El acceso está limitado a la clase contenedora o a las derivadas de la clase contenedora dentro del proyecto actual. Se puede acceder a una clase o miembro private protected solo en la clase que lo declaró o sus derivadas siempre dentro del mismo proyecto.

Desde donde se llama	public	protected	internal	protected internal	private protected	private
Dentro de la clase	✓	✓	✓	✓	✓	✓
Clase derivada (mismo proyecto)	✓	✓	✓	✓	✓	✗
Clase no derivada (mismo proyecto)	✓	✗	✓	✓	✗	✗
Clase derivada (proyecto diferente)	✓	✓	✗	✓	✗	✗
Clase no derivada (proyecto diferente)	✓	✗	✗	✗	✗	✗

Marque la/s afirmaciones verdaderas:

✓ Marque la/s afirmaciones verdaderas:

- [ ] Ninguna respuesta.
- [ ] Los objetos se crean en tiempo de compilación.
- [ ] Los objetos son almacenados en el segmento de memoria conocido como "Stack"
- [x] El valor de las constantes es asignado en tiempo de compilación. ✓
- [ ] Dos objetos del mismo tipo pueden tener distintos valores en una misma constante.

Comentarios

** El valor de las constantes no puede cambiar durante la ejecución del programa, por lo tanto es asignado en tiempo de compilación.*

** Los objetos se crean en tiempo de ejecución.*
** Las constantes se comportan como atributos estáticos, por lo tanto hay un sólo valor para todos los objetos de esa clase. Además, no pueden ser modificadas en tiempo de ejecución, ¿cómo cambiarías su valor desde un constructor de un objeto?.*
** Los objetos se almacenan en el segmento Heap, no en el Stack.*

Marque cuales de las siguientes afirmaciones es verdadera:

✓ Marque cuál/es de las siguientes afirmaciones es verdadera:

- [x] La principal función de las clases abstractas es ser la base de una jerarquía de herencia, definiendo un marco de atributos y comportamiento para todas las clases que deriven de ella. ✓
- [ ] Ninguna respuesta.
- [ ] Las clases estáticas no pueden ser instanciadas, pero pueden heredar.
- [ ] Las clases abstractas no pueden ser instanciadas ni heredadas.
- [ ] Los métodos virtuales definidos en una clase abstracta deberán ser sobrescritos obligatoriamente por la clase derivada.
- [ ] Las clases selladas (sealed) no pueden heredar de otras clases.

Comentarios

** Las clases abstractas no pueden ser instanciadas, pero sí pueden ser heredadas. Están pensadas para ser la base de una jerarquía de herencia.*
** Las clases selladas no pueden ser heredadas (ser base), pero sí pueden heredar (ser derivadas).*
** Las clases estáticas no pueden heredar, ni ser heredadas, ni instanciarse.*
** Los métodos virtuales ya tienen una implementación por defecto, por lo tanto no es obligatoria su invalidación en la clase derivada. Los abstractos, por otro lado, no tienen implementación por defecto y deben ser sobrescritos por las clases derivadas.*

(Herencia)Marque cuales de las siguientes afirmaciones representan propósitos de la herencia:

✓ Marque cuál/es de las siguientes afirmaciones representan propósitos de la herencia:

- [ ] Proteger el acceso a la implementación de la clase base.
- [ ] Ninguna respuesta.
- [x] Reutilizar y organizar mejor el código. ✓
- [x] Extender la estructura y el comportamiento de una clase existente. ✓
- [ ] Crear clases más generales a partir de otras más específicas.

Comentarios

La herencia nos permite crear nuevas clases reutilizando, extendiendo o modificando la estructura y el comportamiento de la clase padre. De esta manera podemos crear clases más específicas a partir de otras más generales.

Su propósito es agrupar atributos y comportamiento en común de un conjunto de clases que componen una determinada realidad, reutilizando y organizando mejor el código.

** El mecanismo de herencia por si mismo no impide el acceso a los detalles de la implementación. Eso tiene más que ver con encapsulamiento.*
** A través de la herencia se pueden crear clases más específicas a partir de otras más generales. Y no al revés.*

Marque cuales de las siguientes afirmaciones son verdaderas:

✓ Marque cuál/es de las siguiente afirmaciones son verdaderas:

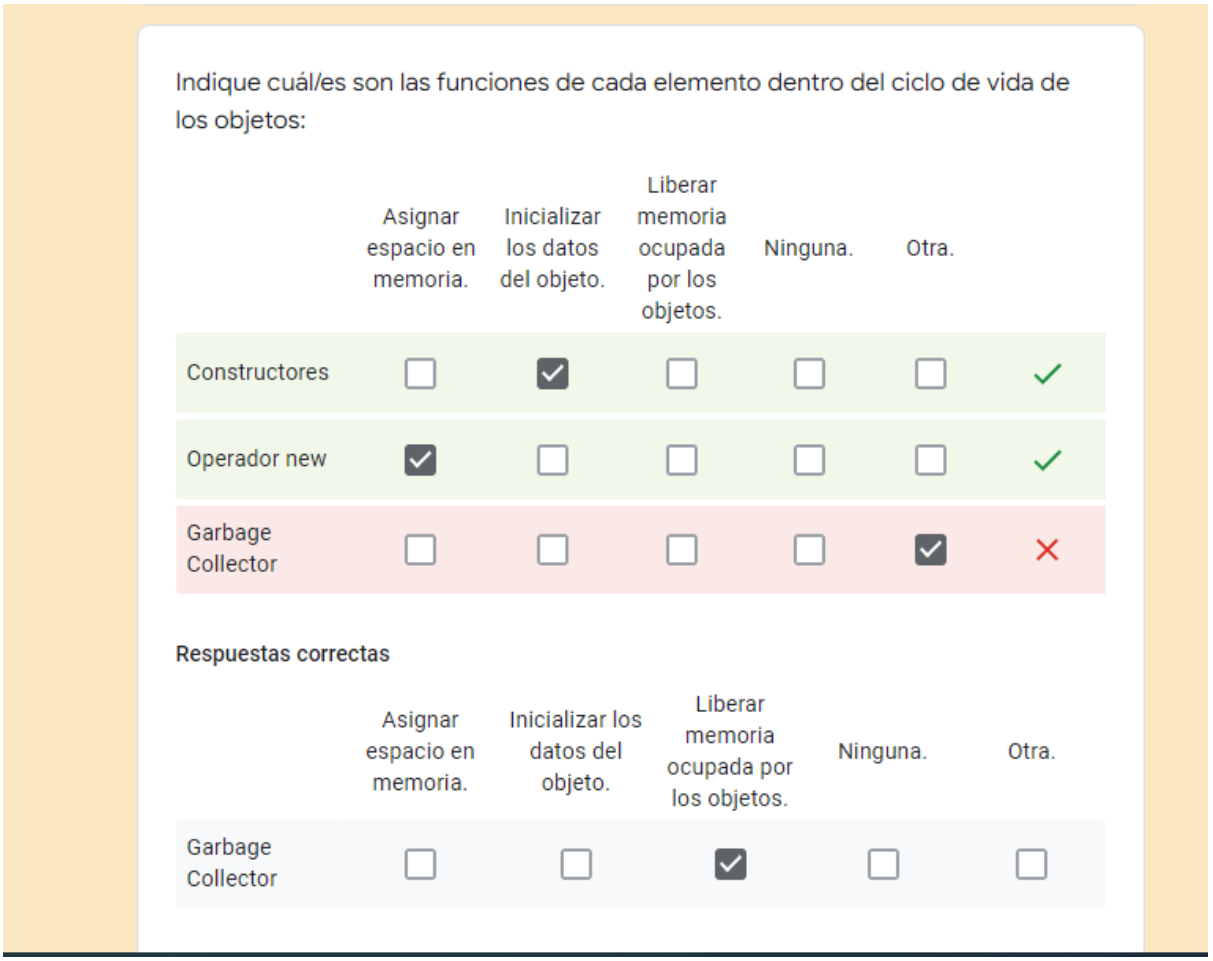
- [ ] Ninguna respuesta.
- [x] El encapsulamiento previene el acceso a los detalles de la implementación y protege el acceso y modificación de los datos del objeto. ✓
- [ ] El modificador internal permitirá que sólo se tenga acceso a los miembros desde la clase donde se declaran.
- [ ] El modificador protected permitirá que sólo se tenga acceso a los miembros desde la clase derivada, no pudiendo acceder desde la clase donde están declarados.
- [ ] Aplicar encapsulamiento es permitir acceso directo e irrestricto para consultar y modificar el estado de un objeto.

Comentarios

El pilar del encapsulamiento consiste en agrupar en un contenedor los datos del objeto junto con los métodos que operan sobre esos datos (Clases). Al mismo tiempo que se ocultan los detalles de la implementación o internos y se protege el acceso a datos. Exponiendo únicamente lo esencial / importante.

** El modificador internal permite acceso desde clases dentro de un mismo ensamblado (unidad de compilación / proyecto).*
** El modificador protected permite acceso desde la misma clase y derivadas.*
** El encapsulamiento protege el acceso a datos y oculta los detalles de la implementación.*

Indique cuales son las funciones de cada elemento dentro del ciclo de vida de los objetos:



Indique cuál/es son las funciones de cada elemento dentro del ciclo de vida de los objetos:

	Asignar espacio en memoria.	Inicializar los datos del objeto.	Liberar memoria ocupada por los objetos.	Ninguna.	Otra.	
Constructores	☐	☑	☐	☐	☐	✓
Operador new	☑	☐	☐	☐	☐	✓
Garbage Collector	☐	☐	☐	☐	☑	✗

Respuestas correctas

	Asignar espacio en memoria.	Inicializar los datos del objeto.	Liberar memoria ocupada por los objetos.	Ninguna.	Otra.
Garbage Collector	☐	☐	☑	☐	☐

(Poli)Marque cual/es de las siguientes afirmaciones son verdaderas:

✗ Marque cuál/es de las siguientes afirmaciones son verdaderas:

- [ ] Ninguna respuesta.
- [ ] El polimorfismo se resuelve en tiempo de compilación.
- [ ] Se aplicar usando el operador new (entre otros).
- [ ] Es la propiedad que permite que objetos de diferentes tipos puedan ser accedidos a través de la misma interfaz, proveyendo una implementación específica dependiendo del tipo en tiempo de ejecución.
- [ ] No todos los objetos de C# son polimórficos.
- [x] Es la propiedad que tienen los objetos de tomar distintas formas (tipos). ✓

Respuesta correcta

- [x] Es la propiedad que tienen los objetos de tomar distintas formas (tipos).
- [x] Es la propiedad que permite que objetos de diferentes tipos puedan ser accedidos a través de la misma interfaz, proveyendo una implementación específica dependiendo del tipo en tiempo de ejecución.

Comentarios

El polimorfismo describe el concepto de que objetos de diferentes tipos pueden ser accedidos a través de la misma interfaz.

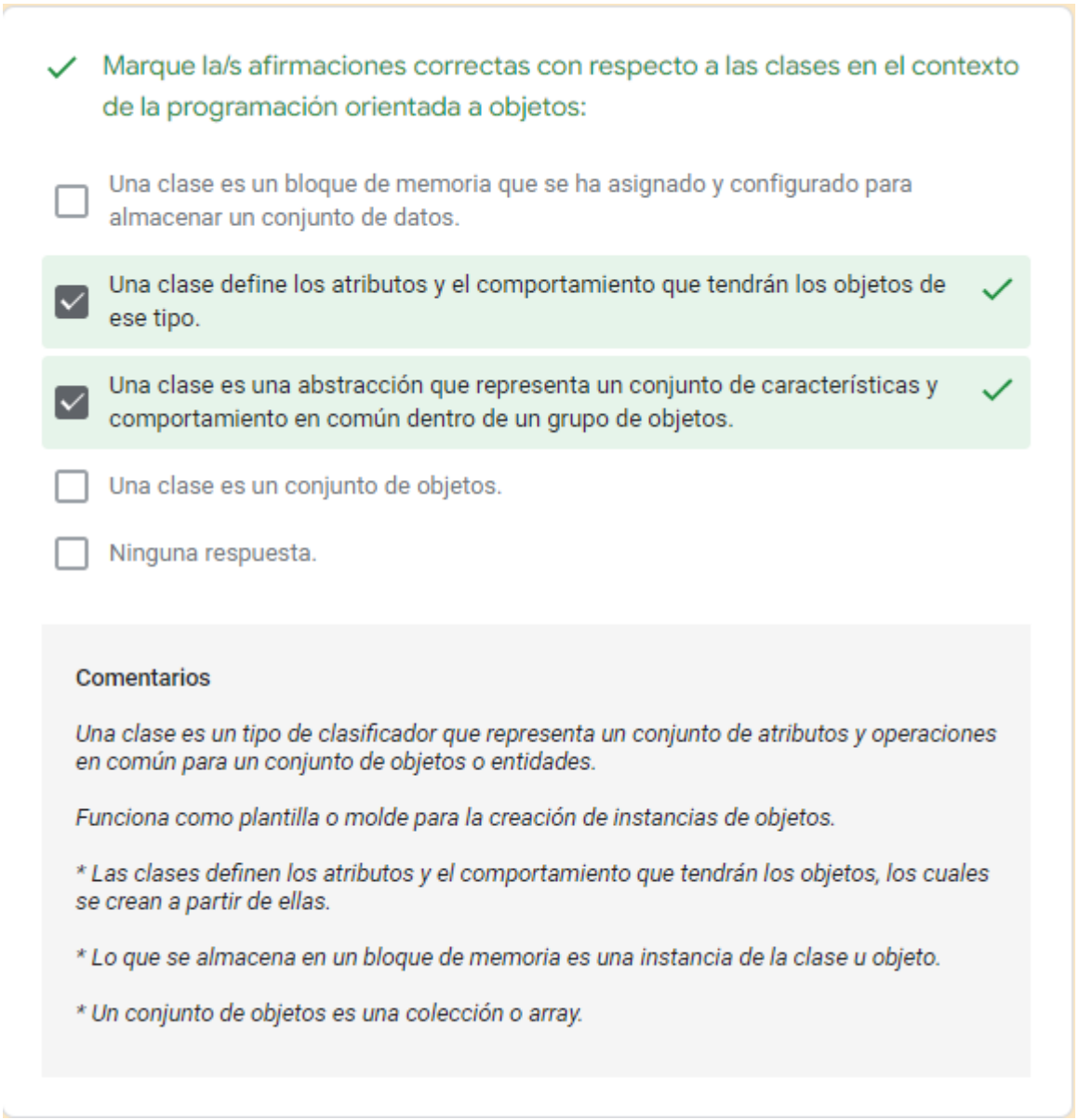
Cada tipo puede proveer su propia implementación de la interfaz.

En otras palabras, en relaciones de herencia, llamando a una misma firma de un método (mismo nombre, parámetros, todo igual), se ejecutará una implementación distinta según el tipo del objeto en tiempo de ejecución.

Es también la propiedad de los objetos de tomar distintas formas (tipos).

** Todos los objetos de C# son polimórficos ya que todos pueden tomar la forma de object (clase de la cual heredan todas las clases de forma implícita).*
** El polimorfismo se resuelve en tiempo de ejecución, ya que los objetos son creados en tiempo de ejecución.*
** El operador new permite declarar un método con el mismo nombre que uno heredado, invalidando el heredado. Pero no permite que se ejecute la implementación correspondiente al tipo del objeto en tiempo de ejecución, sino que se ejecutará la implementación del tipo de la referencia.*

Marque la/s afirmaciones correctas con respecto a las clases en el contexto de la programación orientada a objetos:



Marque la/las afirmaciones verdaderas:

✗ Marque la/s afirmaciones verdaderas:

- [ ] Las colecciones de tipo SortedList están indexadas por el número correspondiente a la posición de sus elementos y por clave / key.
- [ ] Las colecciones de tipo Stack están indexadas por el número correspondiente a la posición de sus elementos.
- [ ] Las colecciones de tipo Queue están indexadas por clave / key.
- [x] Ninguna respuesta. ✗
- [ ] Las colecciones de tipo Queue están indexadas por el número correspondiente a la posición de sus elementos.

Respuesta correcta

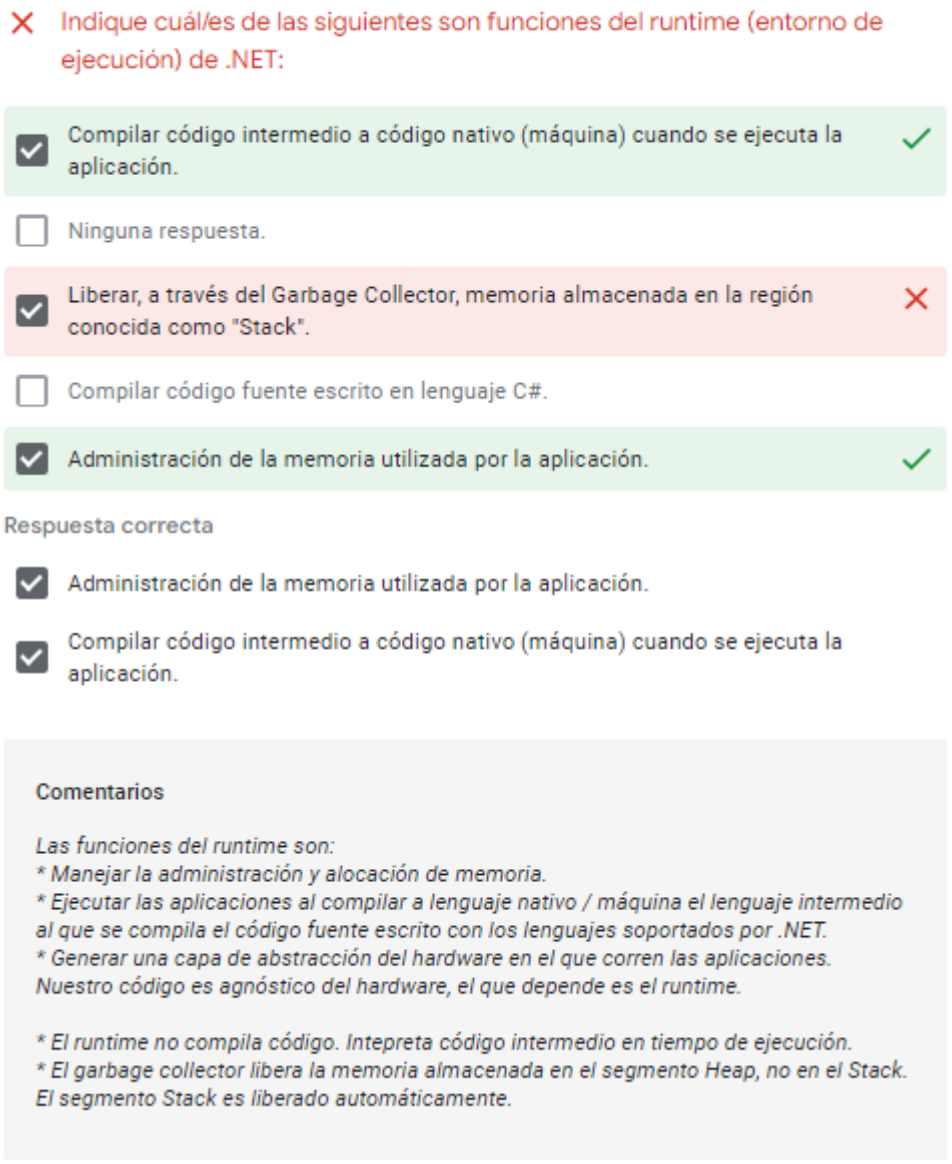
- [x] Las colecciones de tipo SortedList están indexadas por el número correspondiente a la posición de sus elementos y por clave / key.

Comentarios

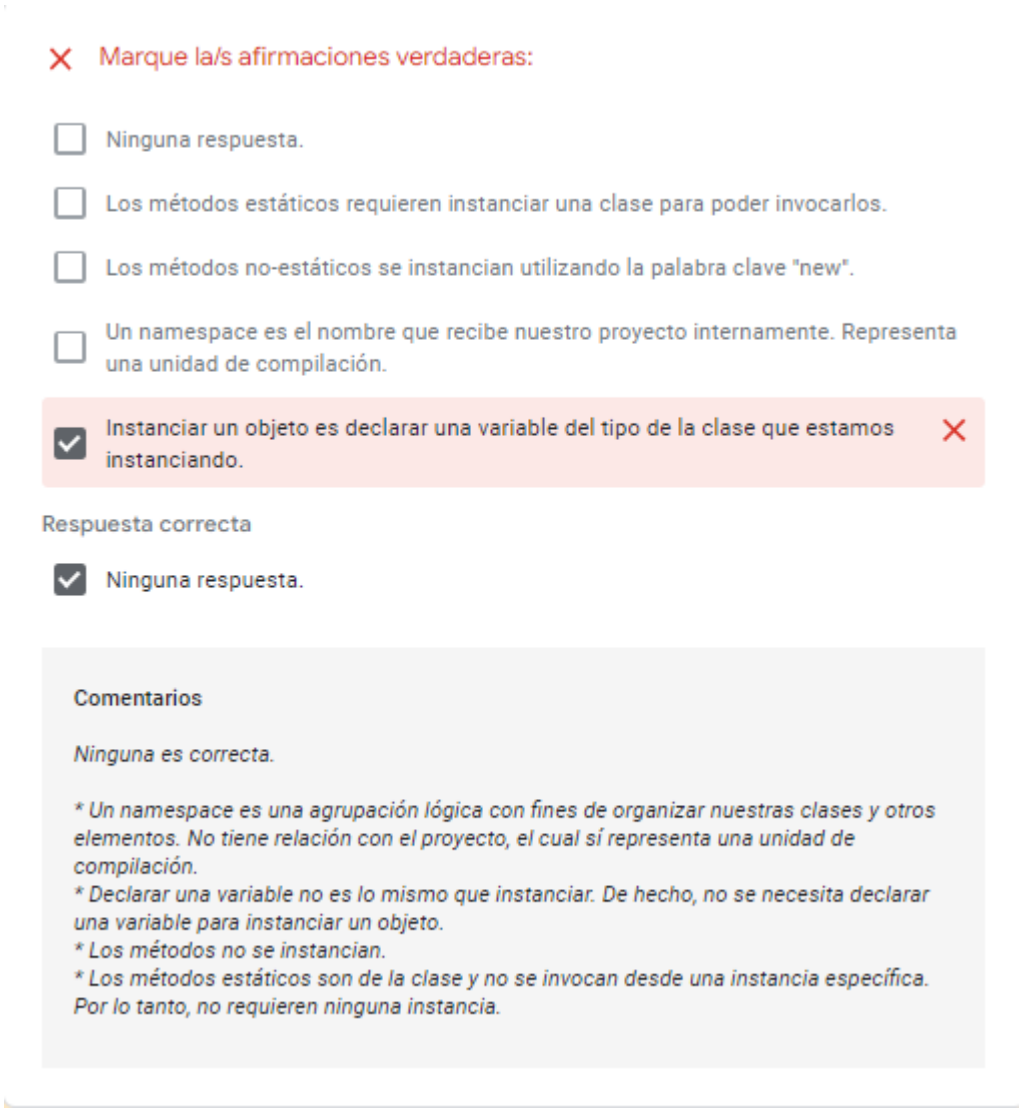
Efectivamente, las colecciones SortedList están conformadas por pares clave-valor que se pueden acceder en base a la posición o por su clave.

Las colecciones tipo Queue y Stack NO están indexadas. Se acceden respectando el orden FIFO y LIFO respectivamente.

Indique cual/es de las siguientes son funciones del runtime(entorno de ejecución) de .NET:



Marque la/s afirmaciones verdaderas:



Marque la/s afirmaciones verdaderas:

✗ Marque la/s afirmaciones verdaderas:

- [x] Un enumerado es un tipo de dato que define un conjunto de constantes numéricas con nombre. ✓
- [ ] Los enumerados sirven para acotar los valores posibles que se pueden asignar a una variable o parámetro.
- [ ] Un enumerado es un descriptor de acceso que define un conjunto de pares clave-valor de tipo <string, int>.
- [ ] Ninguna respuesta.
- [ ] Debe haber una propiedad por cada atributo que tenga la clase, exponiendo siempre TODOS los datos del objeto.
- [x] Las propiedades permiten que una clase exponga una manera segura de exponer y modificar valores almacenados en sus atributos o calculados. ✓

Respuesta correcta

- [x] Un enumerado es un tipo de dato que define un conjunto de constantes numéricas con nombre.
- [x] Los enumerados sirven para acotar los valores posibles que se pueden asignar a una variable o parámetro.
- [x] Las propiedades permiten que una clase exponga una manera segura de exponer y modificar valores almacenados en sus atributos o calculados.

Comentarios

** Siguiendo el principio de encapsulamiento. Se deben exponer únicamente los datos esenciales para quien consume los objetos de la clase.*

Habrá muchos atributos que serán de uso interno y no requieren ser consultados externamente. Habrá otros que serán expuestos a través de un cálculo.

Esto favorece que la clase exponga una interfaz limpia y segura.

** Un enumerado no es un descriptor de acceso (esos son get y set dentro de una propiedad). Tampoco expone pares clave-valor, son constantes numéricas (enteros).*

(Indexadores) Marque la/s afirmaciones correctas respecto a indexadores:

✓ Marque la/s afirmaciones correctas con respecto a indexadores:

- [ ] Indexar es darle al objeto un espacio en memoria.
- [ ] Las clases sólo se pueden indexar por un parámetro numérico.
- [x] En la implementación de un indexador, la palabra clave "value" contendrá el valor a asignar en base al índice proporcionado. ✓
- [x] Indexar es ordenar una serie de datos de acuerdo a un criterio, para facilitar su consulta a través de un índice. ✓
- [ ] Los indexadores sólo se pueden utilizar si la clase tiene declarada una colección como atributo.
- [ ] Ninguna respuesta.

Comentarios

** Darle al objeto un espacio en memoria es parte del proceso de instanciar, no tiene que ver con indexar.*
** Las clases se pueden indexar por múltiples parámetros de cualquier tipo. Depende de la necesidad y del criterio de indexación diseñado para la clase.*
** No es un requisito que la clase tenga una colección o array como atributo para poder ser indexada. De nuevo, depende de la necesidad y del criterio de indexación diseñado.*

(Colecciones)Marque la/s afirmaciones verdaderas:

✓ Marque la/s afirmaciones verdaderas:

- [ ] La diferencia entre colecciones no-genéricas y colecciones genéricas, es que las primeras tienen un tamaño fijo.
- [ ] Las colecciones del tipo Dictionary están indexadas por el número correspondiente a la posición de sus elementos.
- [ ] Ninguna respuesta.
- [x] Las colecciones de tipo SortedList están compuestas por pares clave-valor. ✓
- [ ] Las matrices, a diferencia de los arrays, son dinámicas. Es decir, incrementan y decrementan su tamaño según la demanda en tiempo de ejecución.

Comentarios

** Tanto las colecciones genéricas como las no-genéricas son dinámicas, su tamaño crece y decrece de demanda.*
** Las matrices tienen tamaño fijo. Los arrays son matrices de una sola dimensión.*
** Los diccionarios están compuestos por pares clave-valor y están indexados por la clave o key.*

(WinForm) Marque la/s afirmaciones correctas sobre Windows Forms:

✗ Marque la/s afirmaciones correctas sobre Windows Forms:

- [ ] Los formularios son tipos especiales que no requieren ser instanciados.
- [ ] Ninguna respuesta.
- [ ] El método ShowDialog() visualiza un formulario de forma modal, permitiendo interactuar con otras ventanas.
- [x] Los formularios deben tener obligatoriamente un constructor público sin parámetros. ✗
- [x] El modificador "partial" permite declarar dos clases distintas con el mismo nombre. ✗

Respuesta correcta

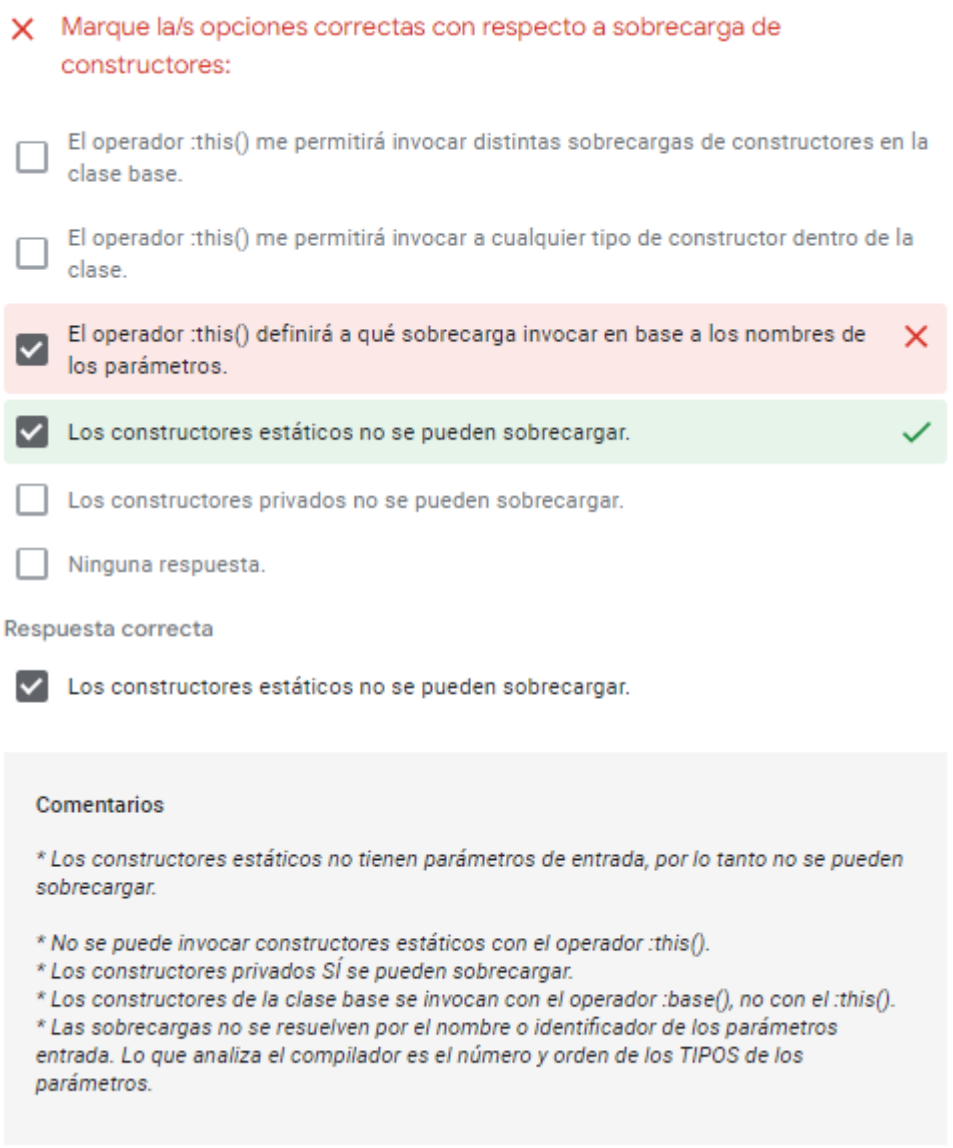
- [x] Ninguna respuesta.

Comentarios

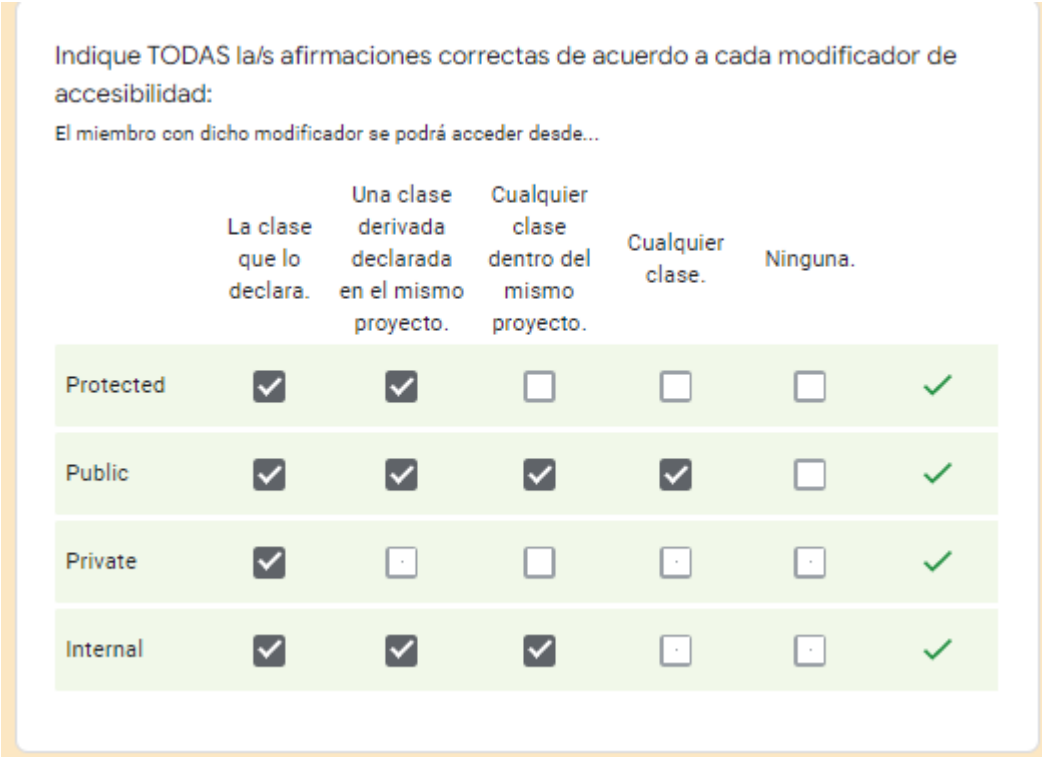
Ninguna es correcta.

** Los formularios son instancias de clases que heredan de Form. Son objetos como cualquier otro.*
** No requieren tener un constructor sin parámetros.*
** El modificador partial permite que la declaración de una misma clase esté separada en dos o más archivos.*
** Efectivamente ShowDialog() es para mostrar el formulario de forma modal, pero las ventanas modales no permiten interactuar con otras ventanas.*

(Sobrecarga constructores)Marque la/s opciones correctas con respecto a sobrecarga de constructores:



(Accesos)Indique TODAS la/s afirmaciones correctas de acuerdo a cada modificador de accesibilidad:



Indique TODAS la/s afirmaciones correctas de acuerdo a cada modificador de accesibilidad:

El miembro con dicho modificador se podrá acceder desde...

	La clase que lo declara.	Una clase derivada declarada en el mismo proyecto.	Cualquier clase dentro del mismo proyecto.	Cualquier clase.	Ninguna.	
Protected	☑	☑	☐	☐	☐	✓
Public	☑	☑	☑	☑	☐	✓
Private	☑	☐	☐	☐	☐	✓
Internal	☑	☑	☑	☐	☐	✓

(Constructores)Marque la/s afirmaciones correctas con respecto a los constructores:

✓ Marque la/s afirmaciones correctas con respecto a los constructores:

- [ ] Los constructores estáticos son los encargados de inicializar los atributos del objeto.
- [x] Si no se declaró de forma explícita un constructor, existirá un constructor público y sin parámetros que será invocado cada vez que se instancie un objeto de esa clase. ✓
- [ ] Ninguna respuesta.
- [ ] El constructor por defecto será invocado por el runtime en el momento de la primera interacción con la clase.
- [ ] Los constructores son los encargados de reservar memoria cuando se instancia un objeto.
- [ ] Los constructores estáticos sólo pueden ser invocados desde métodos estáticos.

Comentarios

Si no se declara de forma explícita un constructor, existirá un constructor por defecto. El constructor por defecto es público y no tiene parámetros de entrada.

** Los constructores inicializan los atributos del objeto. No reservan memoria, esa tarea es del operador new.*
** El constructor por defecto - si no fue reemplazado - se invoca al instanciar un nuevo objeto de esa clase.*
** Los constructores estáticos no pueden ser invocados. La invocación la realiza el runtime en la primera interacción con la clase.*
** Los constructores estáticos son de la clase, no de una instancia específica. Por lo tanto, sólo pueden inicializar atributos de clase (estáticos).*

(Compilacion c#)Marque la/s afirmaciones correctas sobre el proceso de compilación de C#:

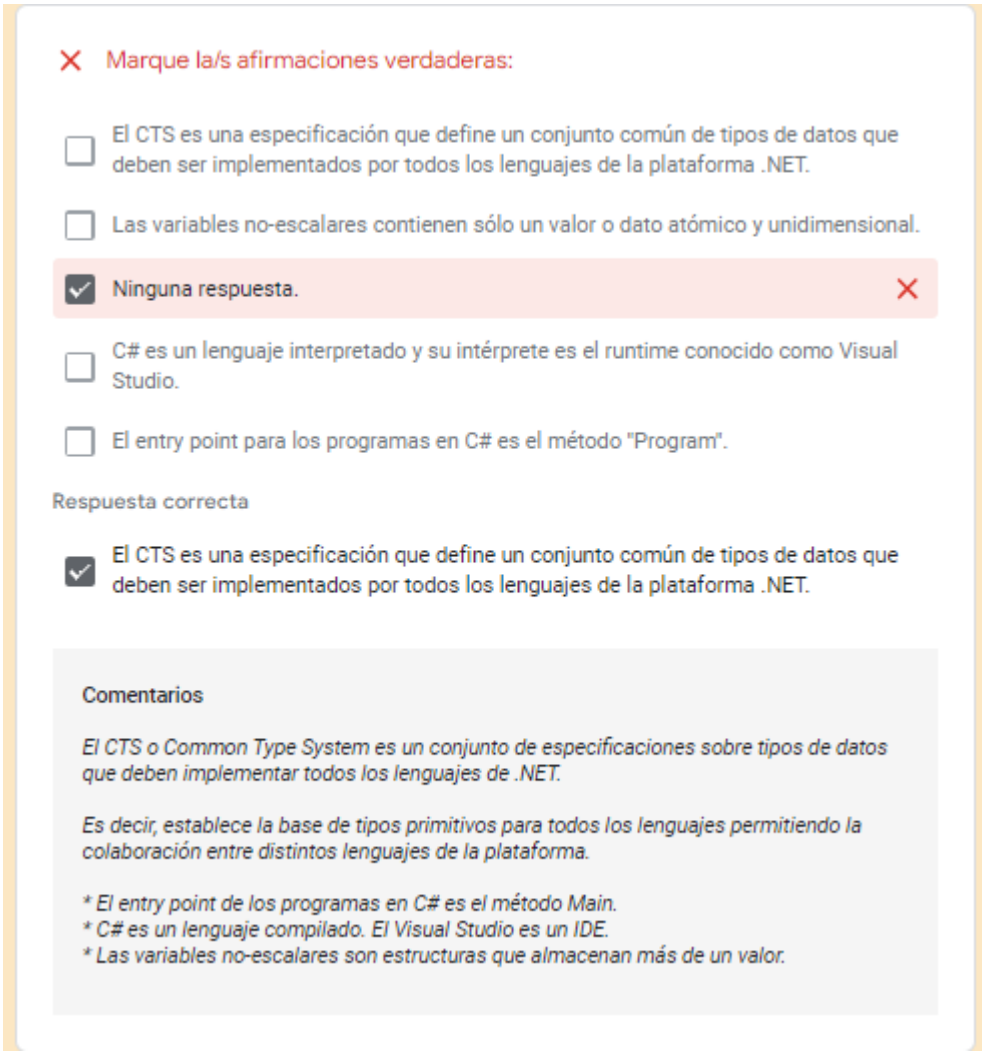
✓ Marque la/s afirmaciones correctas sobre el proceso de compilación de C#:

- [ ] Ninguna respuesta.
- [x] El compilador de C# traducirá el código fuente a código intermedio en tiempo de compilación, necesitando la intervención de un intérprete en tiempo de ejecución. ✓
- [ ] El código en lenguaje C# será interpretado por el runtime en tiempo de ejecución.
- [ ] El Visual Studio se encargará de interpretar el código intermedio generado por el entorno de ejecución.
- [ ] El compilador de C# traducirá el código fuente a lenguaje máquina, permitiendo su ejecución sin intermediarios.

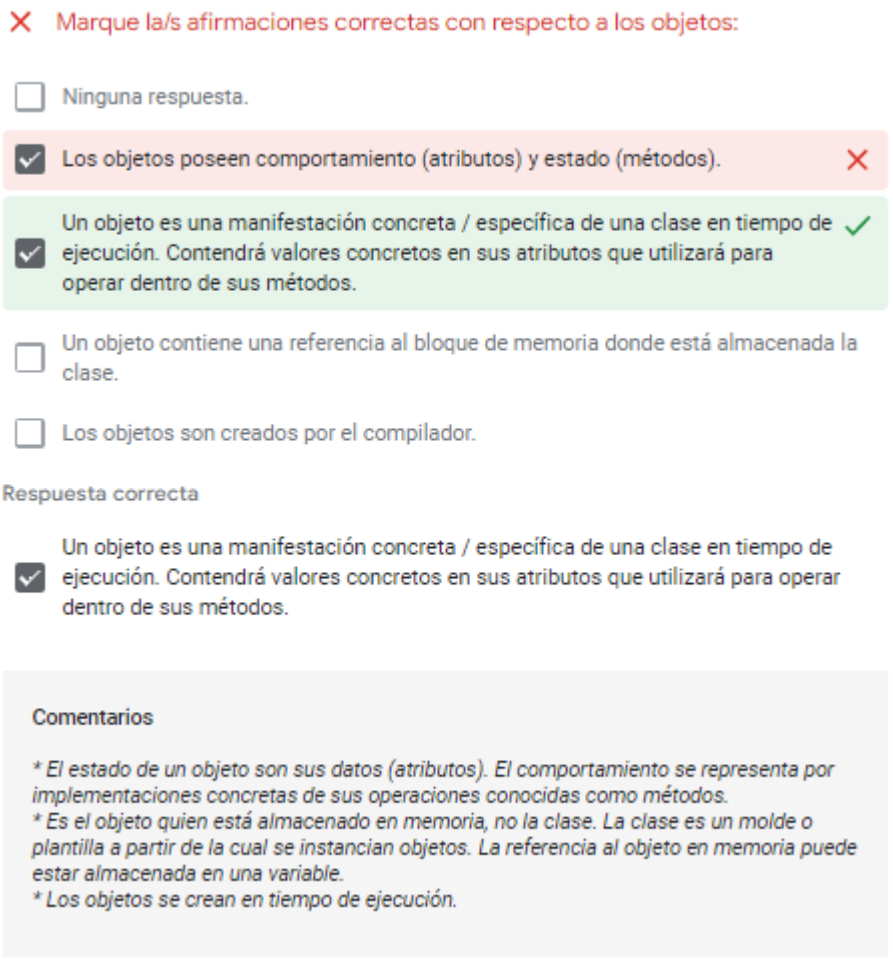
Comentarios

** Lo que interpreta el runtime en tiempo de ejecución es código intermedio o bytecode, no C#.*
** El Visual Studio es un IDE, no un interprete. El código intermedio es generado por el compilador, no por el runtime.*
** El compilador de C# traduce el código fuente a código intermedio, el cual deberá ser interpretado en tiempo de ejecución por el runtime.*

(Common type system)Marque la/s afirmaciones verdaderas:

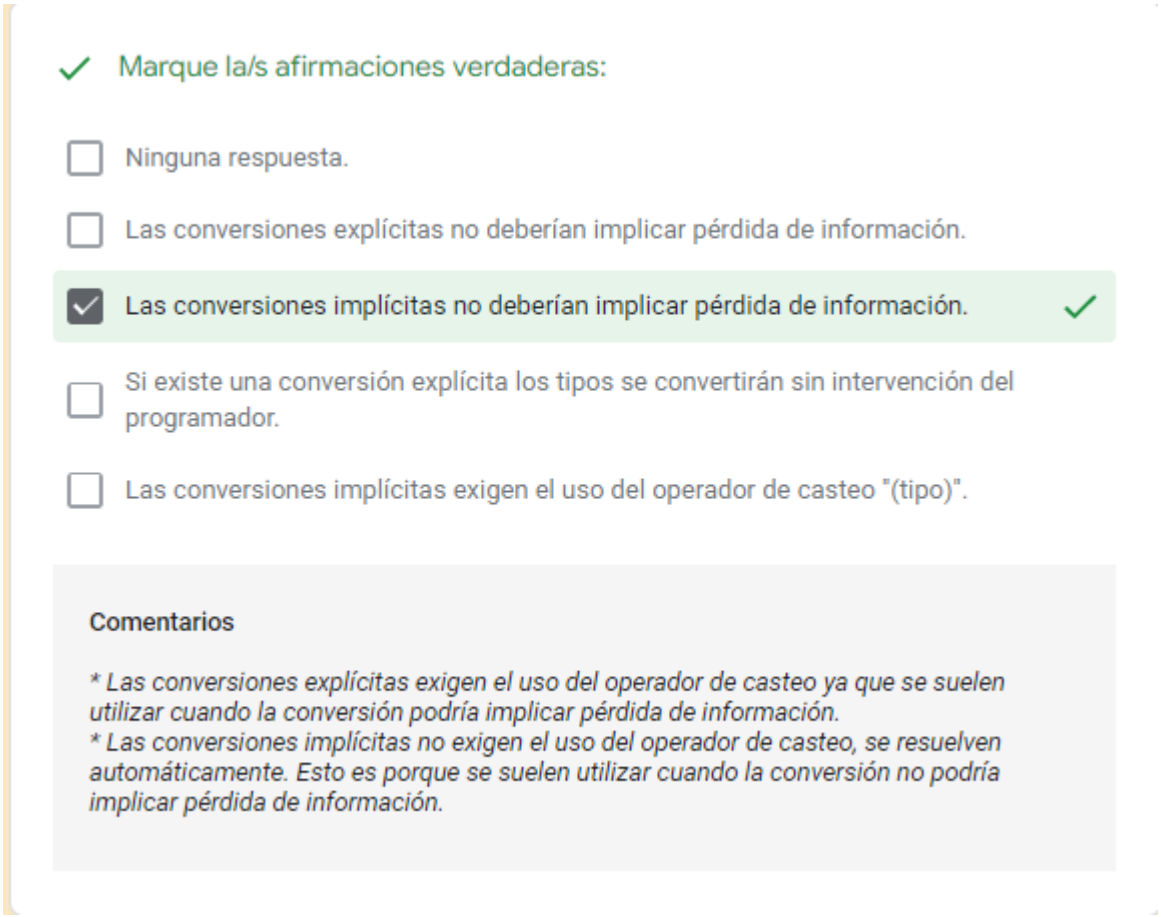


(Objetos)Marque la/s afirmaciones correctas con respecto a los objetos:

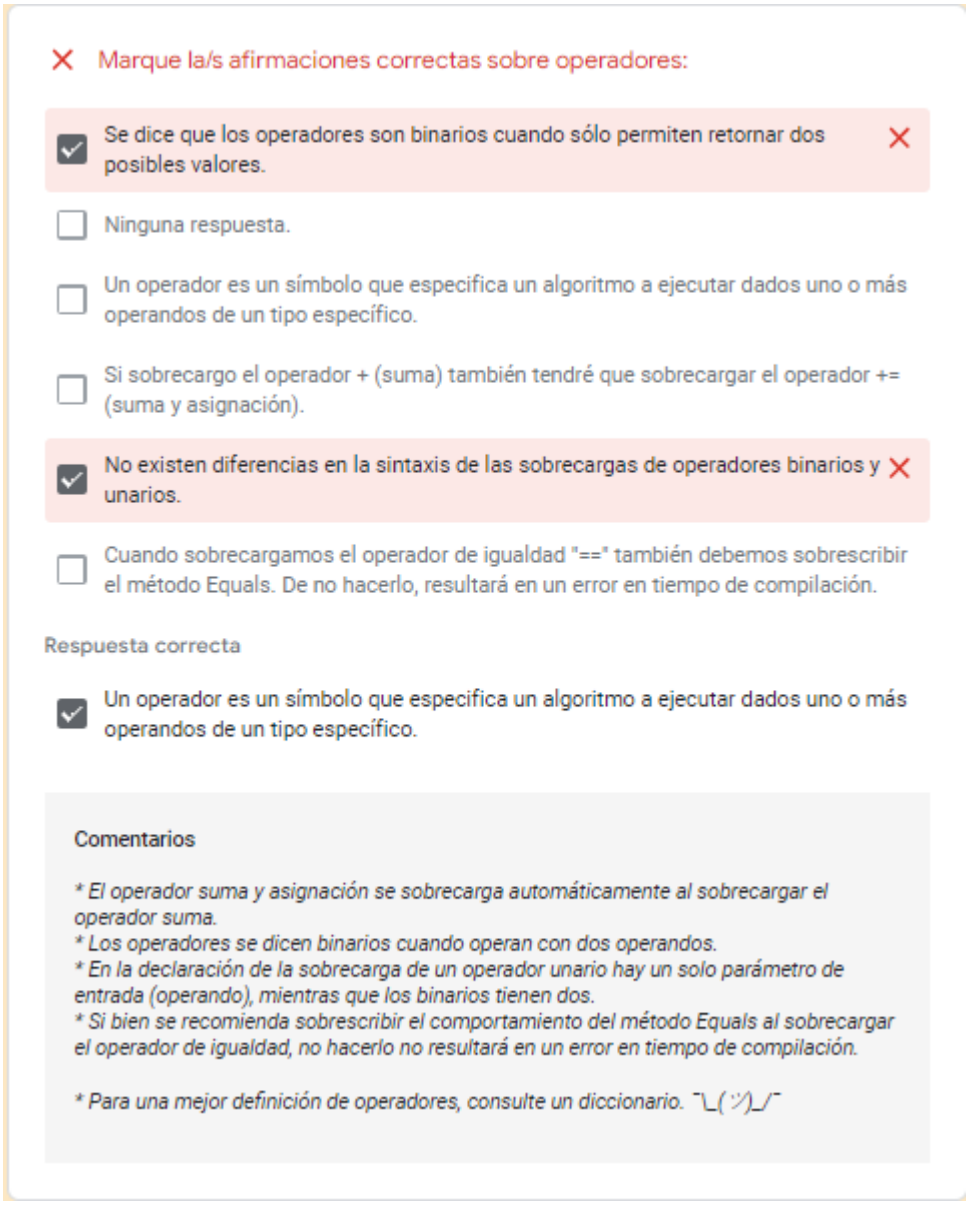


Un objeto es un bloque de memoria que se ha asignado y configurado de acuerdo a las especificaciones de una clase.

(Conversiones)Marque la/s afirmaciones verdaderas:



(Operadores)Marque la/s afirmaciones correctas sobre operadores:



✕ Marque la/s afirmaciones correctas sobre operadores:

- [x] Se dice que los operadores son binarios cuando sólo permiten retornar dos posibles valores. ✕
- [ ] Ninguna respuesta.
- [ ] Un operador es un símbolo que especifica un algoritmo a ejecutar dados uno o más operandos de un tipo específico.
- [ ] Si sobrecargo el operador + (suma) también tendré que sobrecargar el operador += (suma y asignación).
- [x] No existen diferencias en la sintaxis de las sobrecargas de operadores binarios y unarios. ✕
- [ ] Cuando sobrecargamos el operador de igualdad "==" también debemos sobrescribir el método Equals. De no hacerlo, resultará en un error en tiempo de compilación.

Respuesta correcta

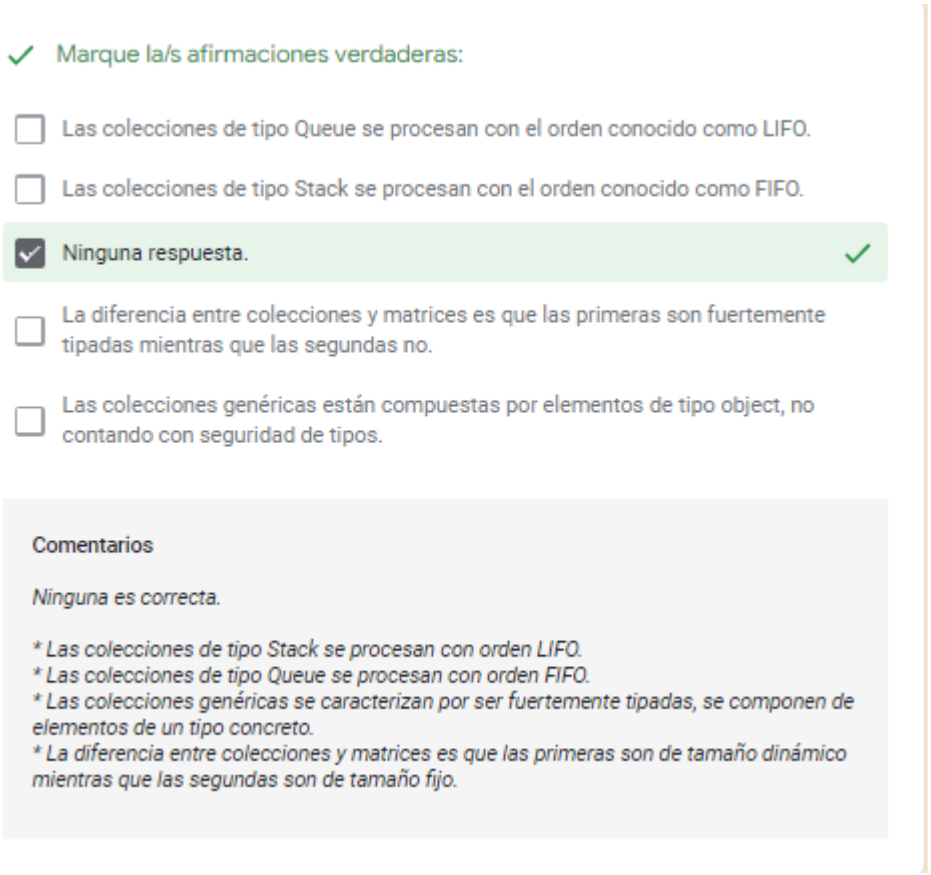
- [x] Un operador es un símbolo que especifica un algoritmo a ejecutar dados uno o más operandos de un tipo específico.

Comentarios

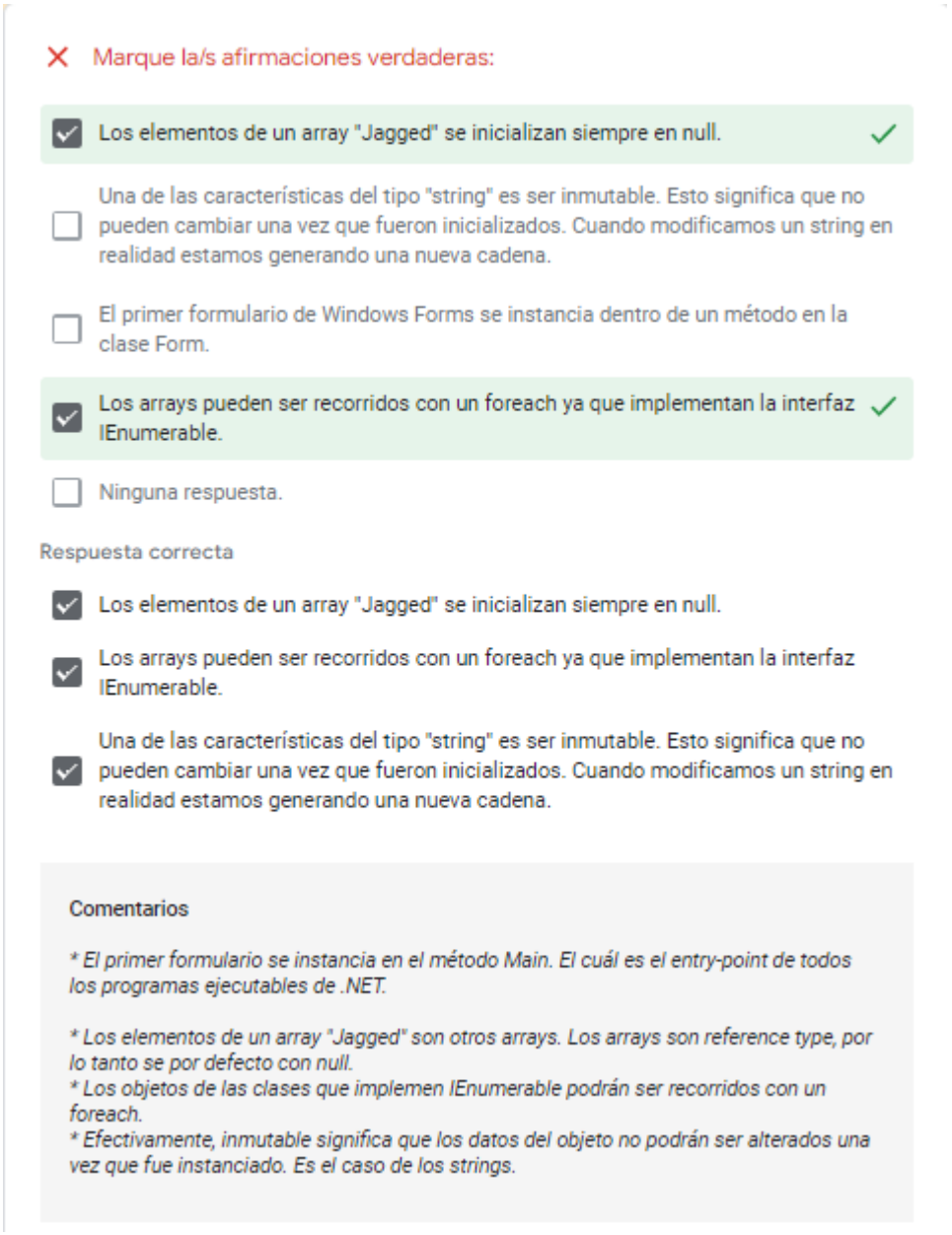
** El operador suma y asignación se sobrecarga automáticamente al sobrecargar el operador suma.*
** Los operadores se dicen binarios cuando operan con dos operandos.*
** En la declaración de la sobrecarga de un operador unario hay un solo parámetro de entrada (operando), mientras que los binarios tienen dos.*
** Si bien se recomienda sobrescribir el comportamiento del método Equals al sobrecargar el operador de igualdad, no hacerlo no resultará en un error en tiempo de compilación.*

** Para una mejor definición de operadores, consulte un diccionario.* ¯_(ツ)_/¯

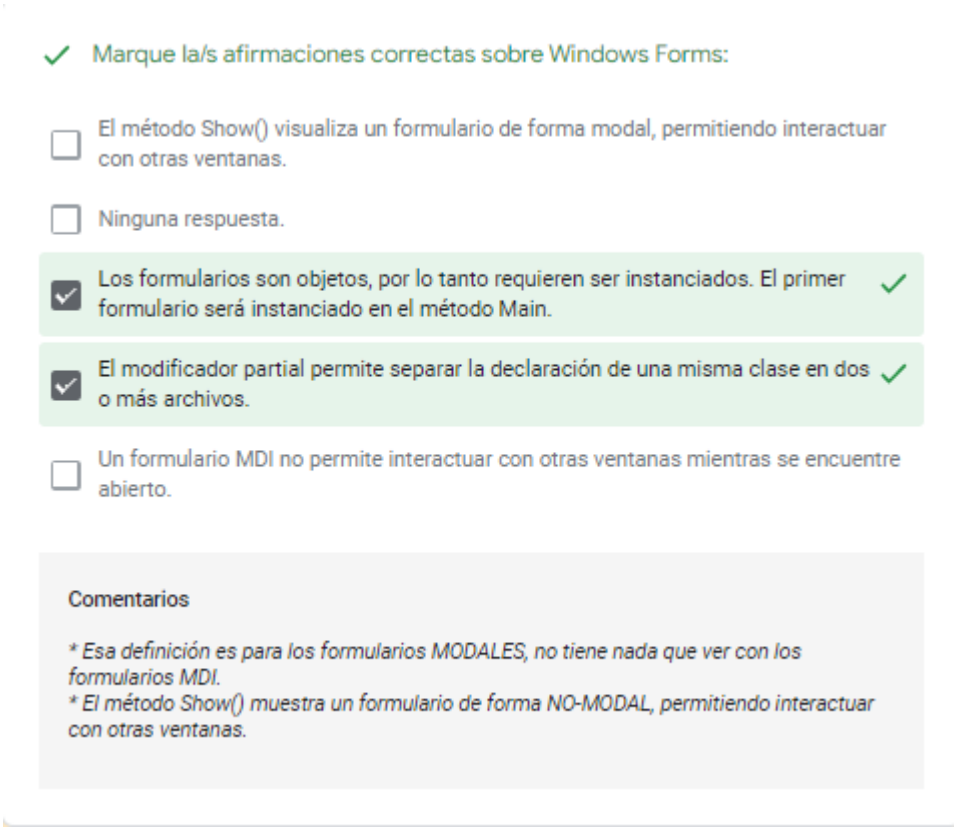
(Colecciones)Marque la/s afirmaciones verdaderas:



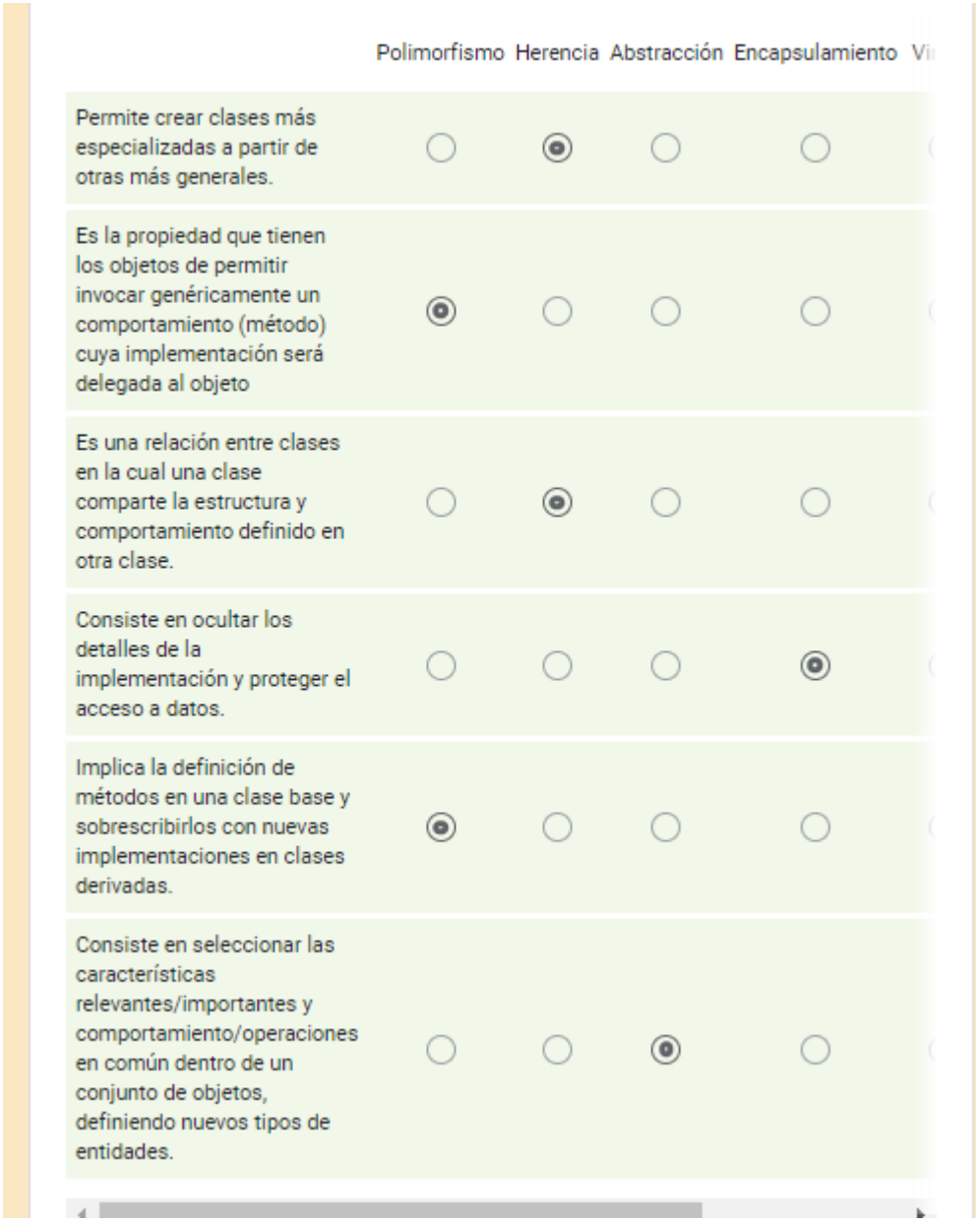
(Array, string)Marque la/s afirmaciones verdaderas:



(Windows Forms)Marque la/s afirmaciones correctas sobre Windows Forms:



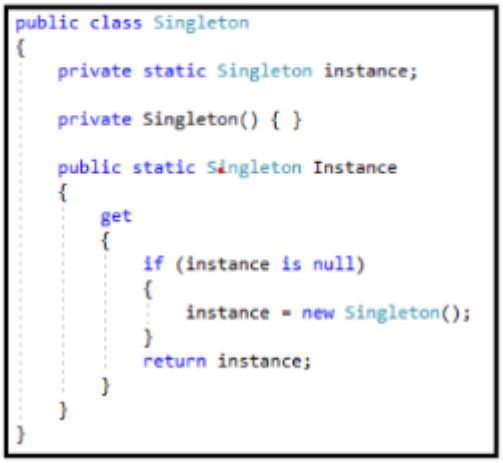
Indique el nombre del pilar de la programación orientada a objetos correspondiente a la descripción:



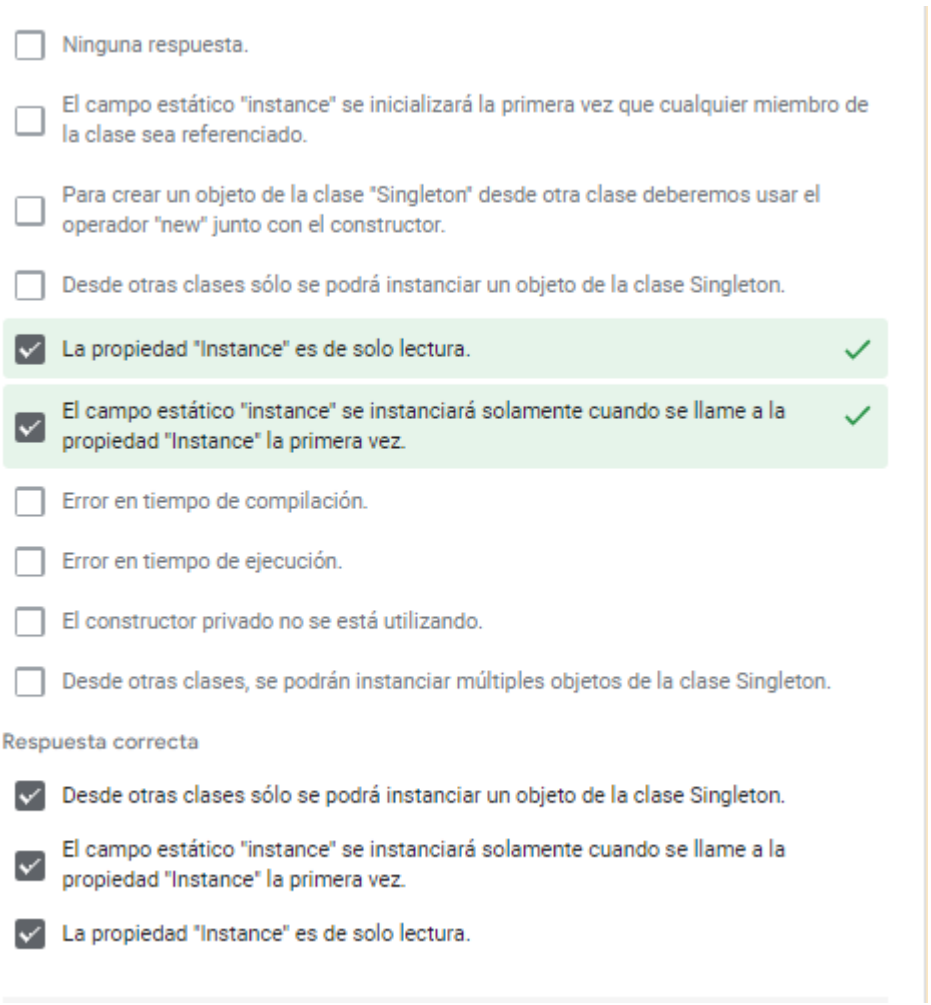
	Polimorfismo	Herencia	Abstracción	Encapsulamiento	Vi
Permite crear clases más especializadas a partir de otras más generales.	○	◉	○	○	
Es la propiedad que tienen los objetos de permitir invocar genéricamente un comportamiento (método) cuya implementación será delegada al objeto	◉	○	○	○	
Es una relación entre clases en la cual una clase comparte la estructura y comportamiento definido en otra clase.	○	◉	○	○	
Consiste en ocultar los detalles de la implementación y proteger el acceso a datos.	○	○	○	◉	
Implica la definición de métodos en una clase base y sobrescribirlos con nuevas implementaciones en clases derivadas.	◉	○	○	○	
Consiste en seleccionar las características relevantes/importantes y comportamiento/operaciones en común dentro de un conjunto de objetos, definiendo nuevos tipos de entidades.	○	○	◉	○	

El siguiente código corresponde con un patrón de diseño llamado Singleton. Interprete el código e indique cuál/es de las siguientes afirmaciones es correcta:

✕ El siguiente código corresponde con un patrón de diseño llamado Singleton. Interprete el código e indique cuál/es de las siguientes afirmaciones es correcta:



```
public class Singleton
{
    private static Singleton instance;

    private Singleton() { }

    public static Singleton Instance
    {
        get
        {
            if (instance is null)
            {
                instance = new Singleton();
            }
            return instance;
        }
    }
}
```



¿Qué texto de salida muestra el programa de consola de las imágenes?

```
public class Numero
{
    private int numero;

    private Numero(int numero)
    {
        this.numero = numero;
    }

    public int NumeroLoco
    {
        get { return this.numero; }
        set { this.numero = value; }
    }

    public static implicit operator Numero(int numero)
    {
        return new Numero(numero);
    }
}
```

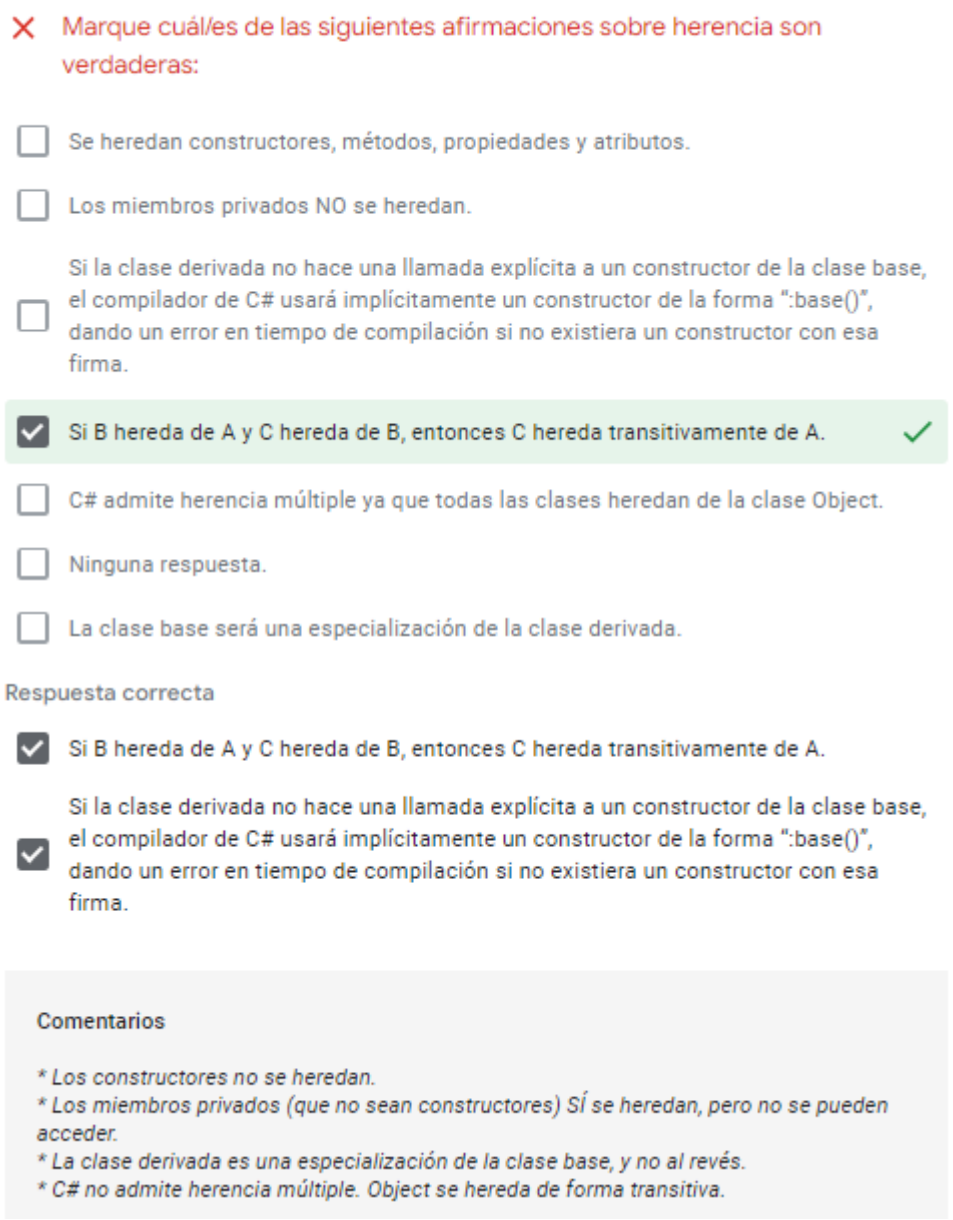
```
static void Main(string[] args)
{
    int numeroUno = 1;
    int numeroDos = 2;
    Numero numeroTres = 3;

    MetodoLoco(numeroUno);
    MetodoLoco(out numeroDos);
    MetodoLoco(numeroTres);

    Console.WriteLine("{0} - {1} - {2}", numeroUno, numeroDos, numeroTres.NumeroLoco);
    Console.ReadKey();
}

public static void MetodoLoco(int numero)
{
    numero = 14;
}

public static void MetodoLoco(out int numero)
{
    numero = 14;
}

public static void MetodoLoco(Numero numero)
{
    numero.NumeroLoco = 14;
}
```

- ○ 14 - 14 - 14
- ○ 1 - 14 - 3
- ◉ 1 - 14 - 14 ✓
- ○ 14 - 2 - 3
- ○ 14 - 2 - 14
- ○ Ninguna respuesta.
- ○ 1 - 2 - 14
- ○ 14 - 14 - 3

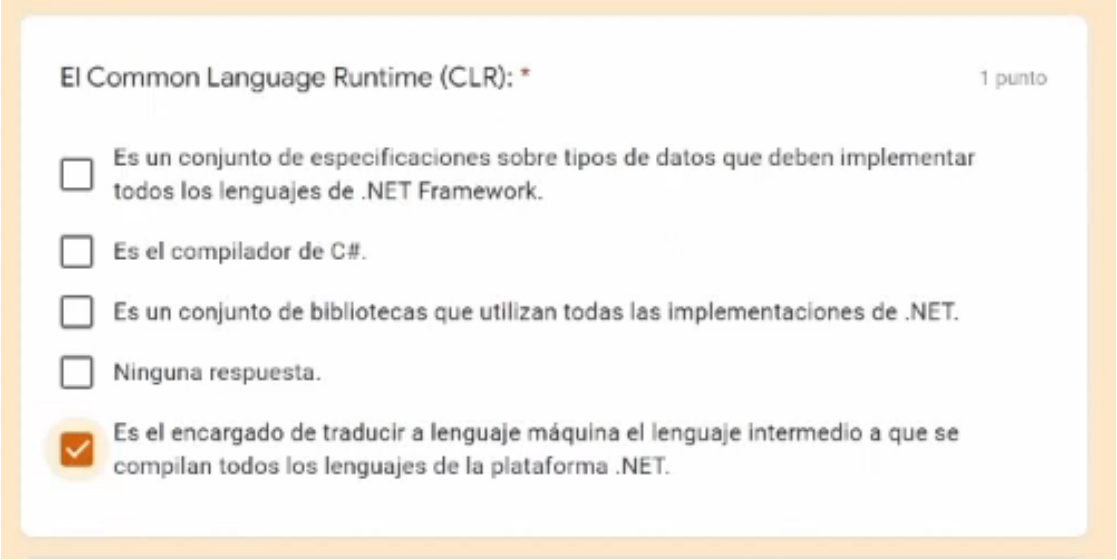
Comentarios

** El tipo int es un value type, al pasarse como argumento a un método se realizará una copia del valor y cualquier modificación al mismo dentro del método NO se verá reflejada en el ámbito de llamada.*
** Los value type que se pasen a un parámetro con el modificador "out" se comportarán como argumentos de tipo reference, lo que tiene como consecuencia que cualquier modificación al mismo dentro del método se verá reflejada dentro del ámbito de llamada.*
** Los objetos son reference type, cualquier modificación de los mismos dentro de un método se verá reflejada en el ámbito de llamada.*

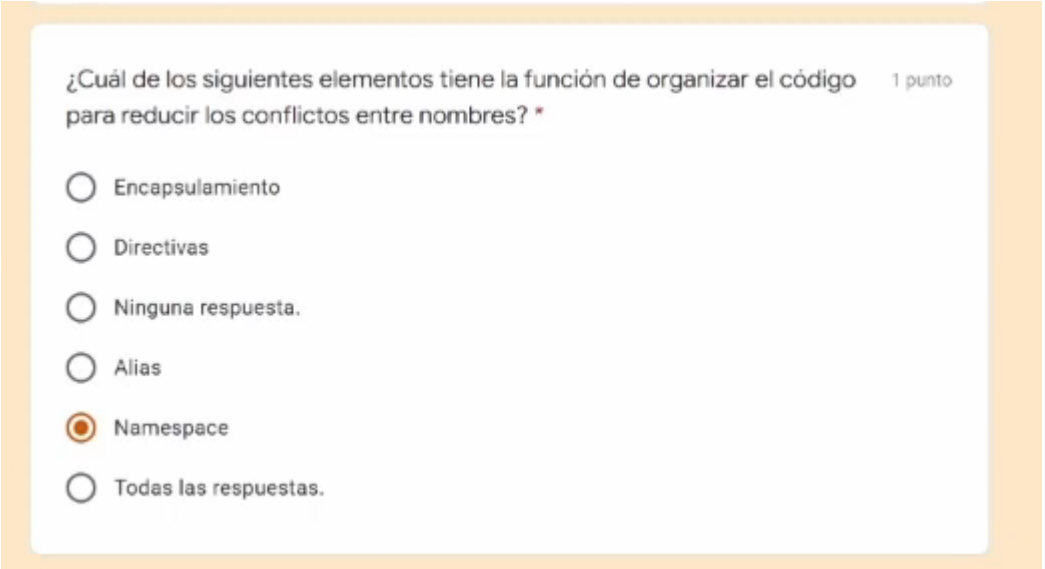
(Herencia)Marque cuál/es de las siguientes afirmaciones sobre herencia son verdaderas:



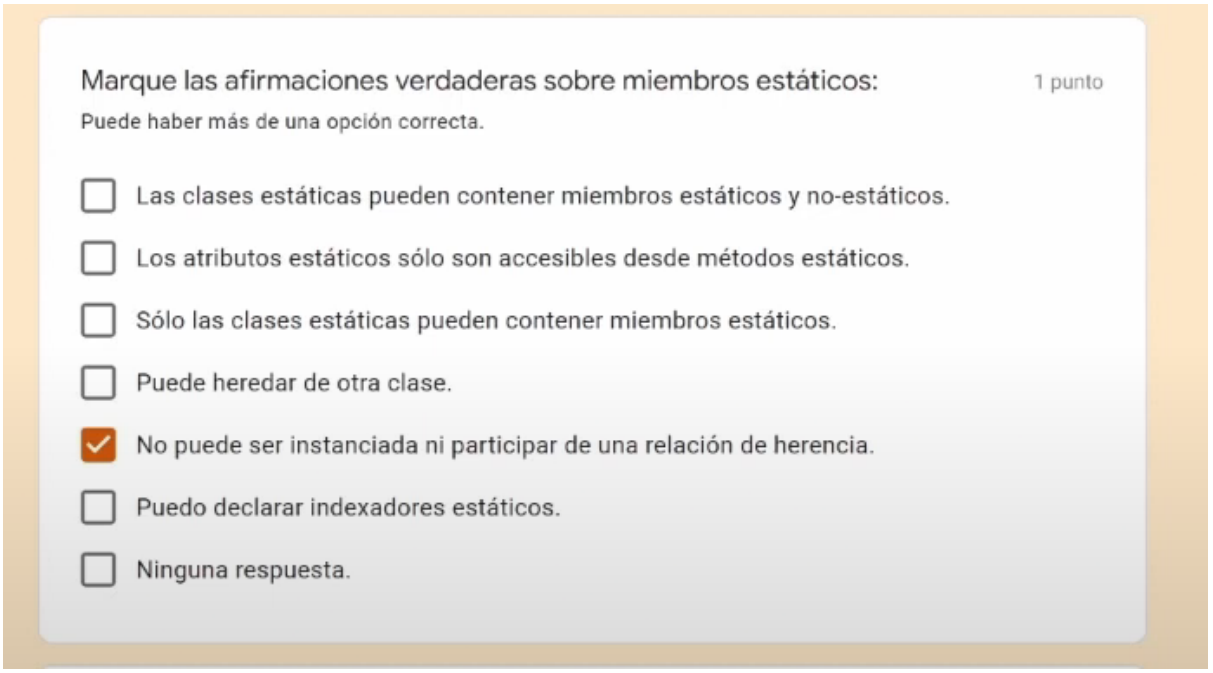
El Common Language Runtime (CLR):



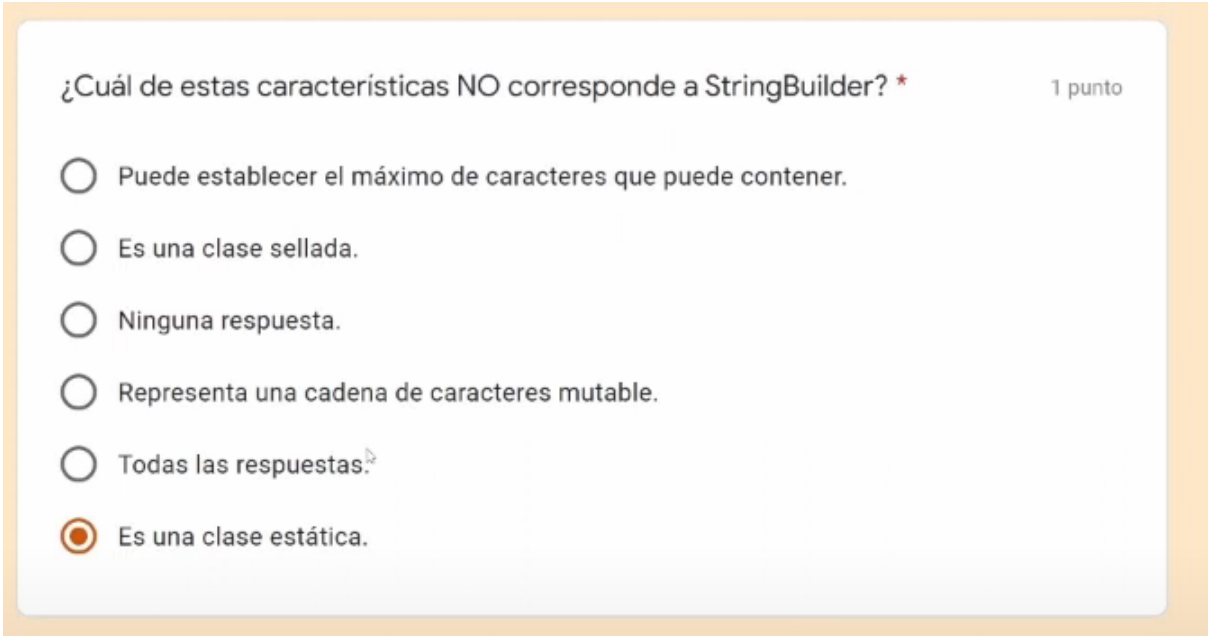
¿Cuál de los siguientes elementos tiene la función de organizar el código para reducir los conflictos entre nombres?



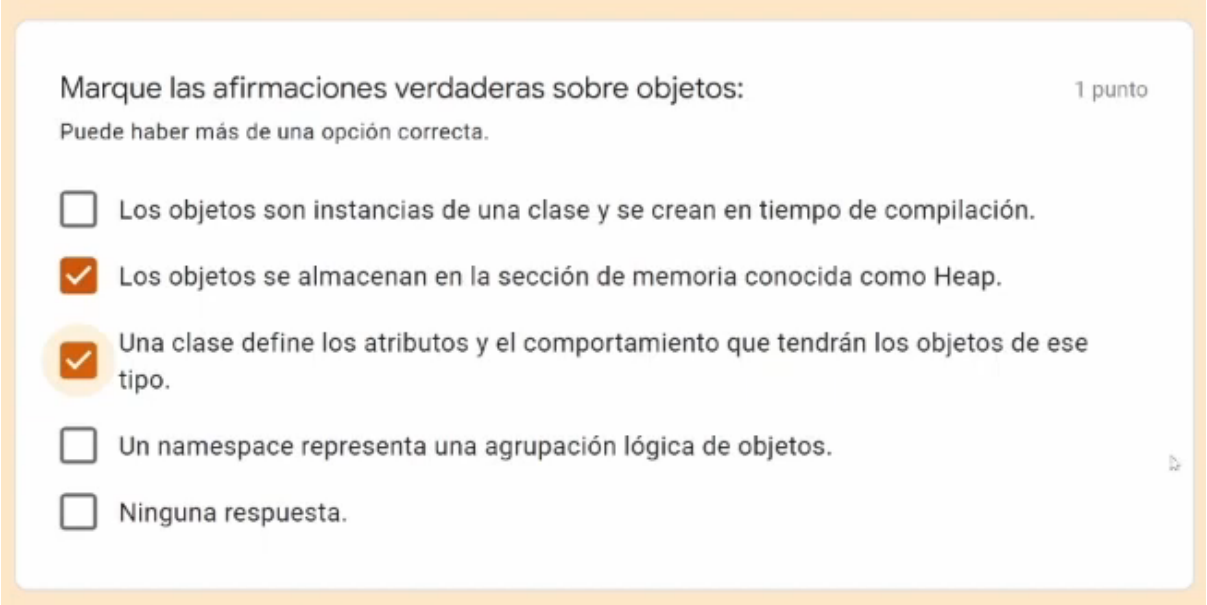
(Miembros estáticos)Marque las afirmaciones verdaderas sobre miembros estáticos:



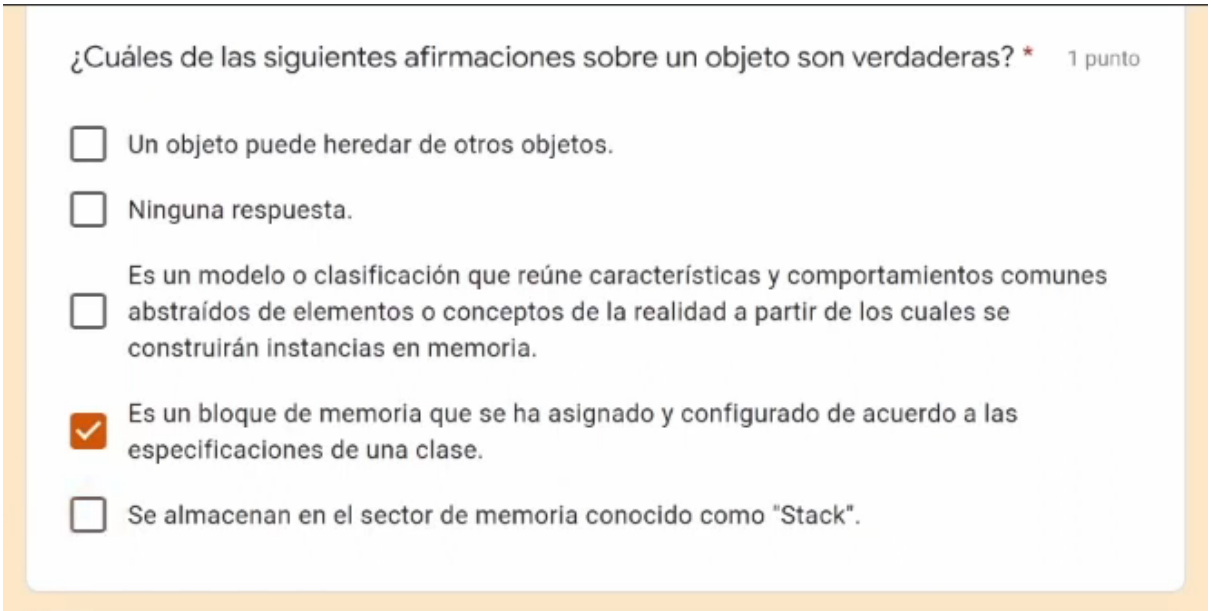
(StringBuilder) ¿Cuál de estas características NO corresponde a StringBuilder?



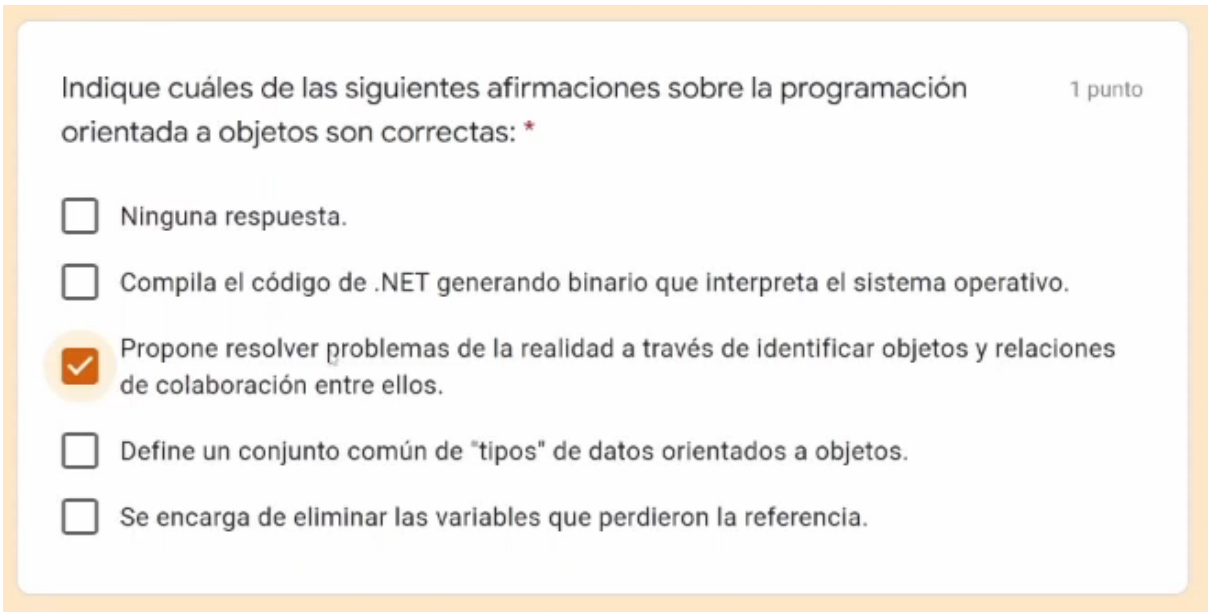
(Objetos) Marque las afirmaciones verdaderas sobre objetos:



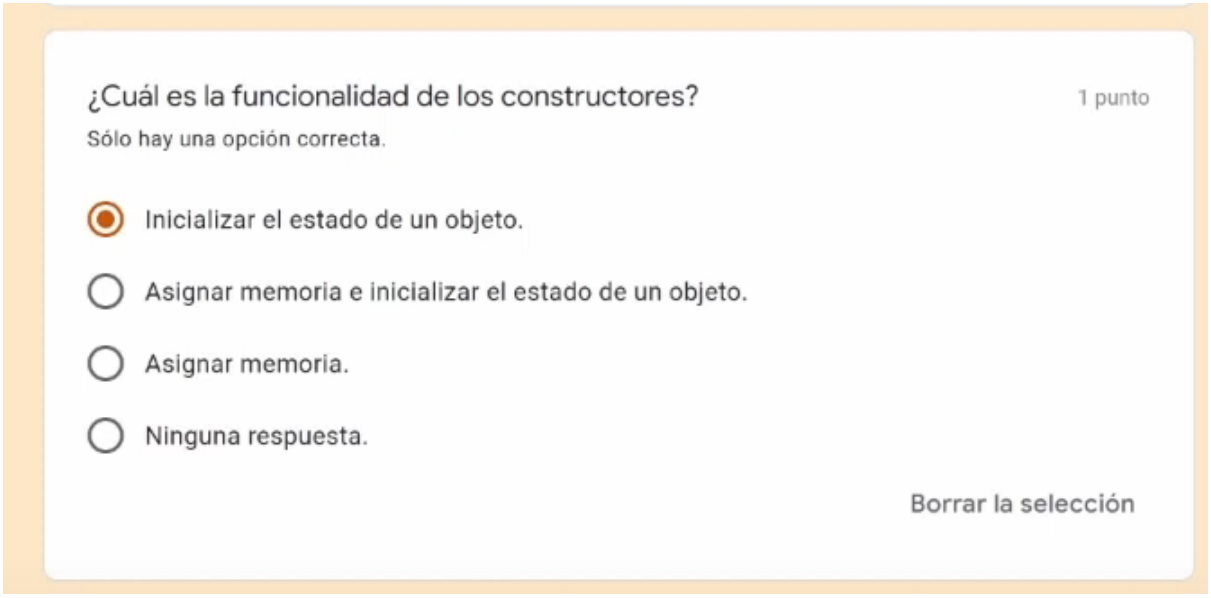
(Objetos)¿Cuáles de las siguientes afirmaciones sobre un objeto son verdaderas?



(POO)cuáles de las afirmaciones sobre la programación orientada a objetos son correctas:



(Constructores)¿Cuál es la funcionalidad de los constructores?



(Constructores privados, públicos, estáticos)

Si tengo una clase con solamente tres constructores: uno estático, uno privado y uno público. Indique la afirmación correcta: * 1 punto

- ○ El constructor privado sólo podrá alterar elementos que sean privados o protegidos.
- ○ Ninguna respuesta.
- ◉ El constructor estático sólo podrá operar sobre otros miembros de la clase que también sean estáticos.
- ○ El constructor estático llamará al constructor privado.
- ○ Al instanciar el primer objeto de ese tipo, el primer constructor por el que pasará será el constructor público.
- ○ Todas las respuestas.
- ○ Será obligatorio hacer una llamada al constructor privado mediante alguno de los otros constructores, dando un error si no se hace.

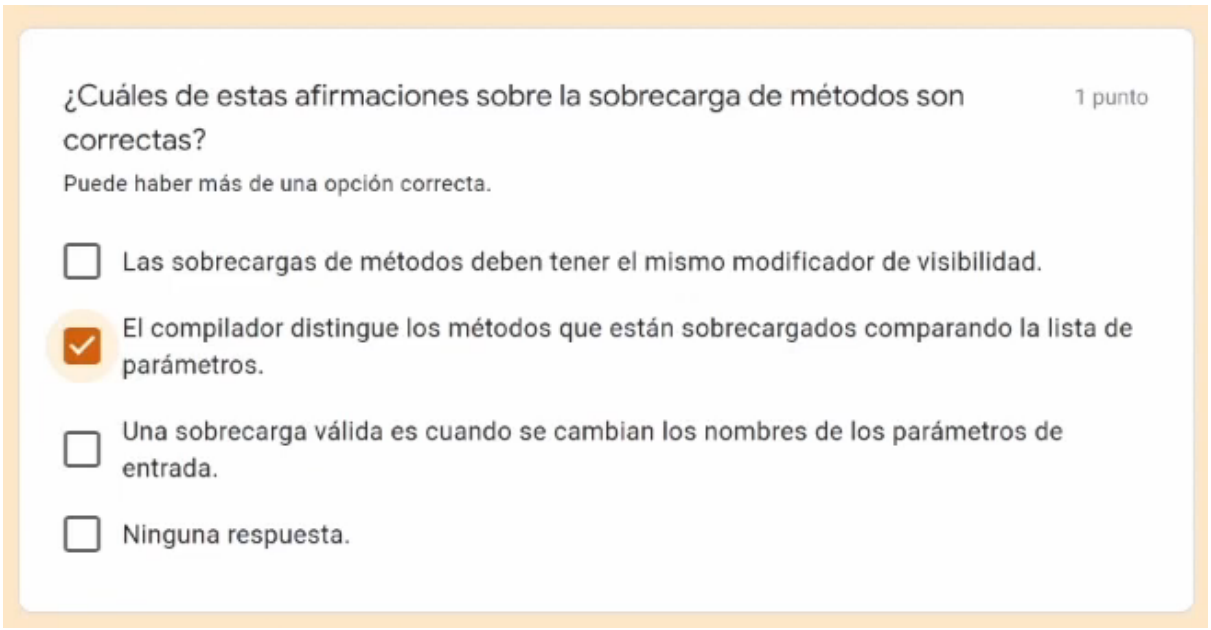
(Herencia constructores) Indique el o los errores del siguiente código:

Indique el o los errores del siguiente código: * 1 punto

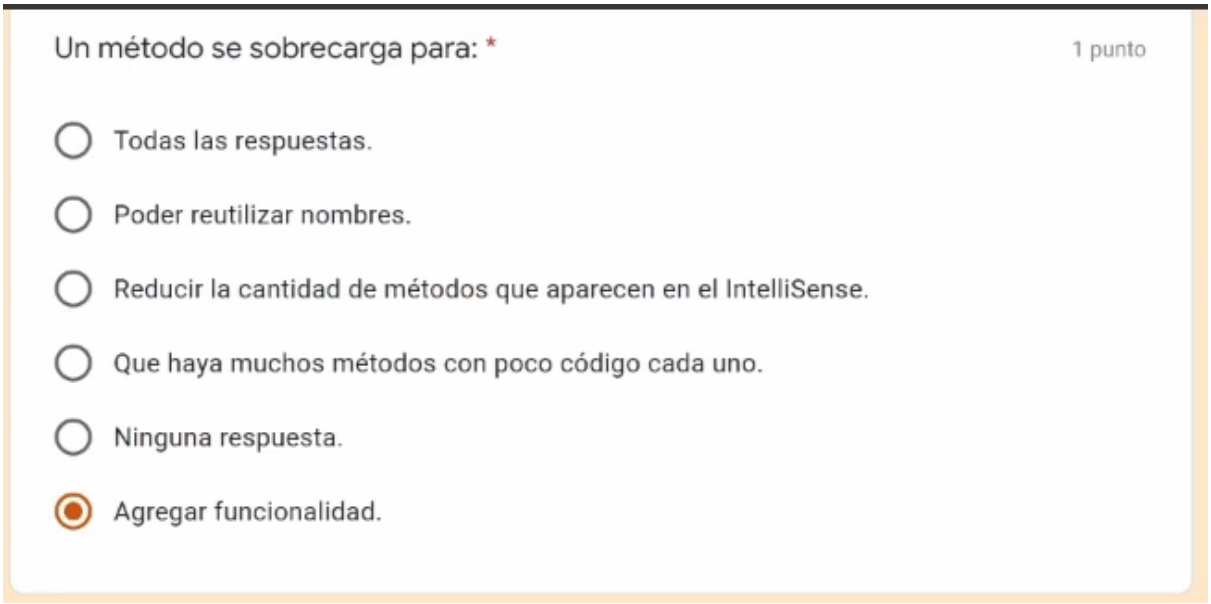
```
class PruebaB
{
public PruebaB():base() {}
static PruebaB(){}
public PruebaB(int par) : this() { }
}
```

- ☑ No se puede llamar a :base en esta clase.
- ☐ No puede haber 3 métodos con el mismo nombre.
- ☐ Ninguna respuesta.
- ☐ No se puede llamar a :this en esta clase.
- ☐ El constructor estático no puede existir si la clase no es estática.

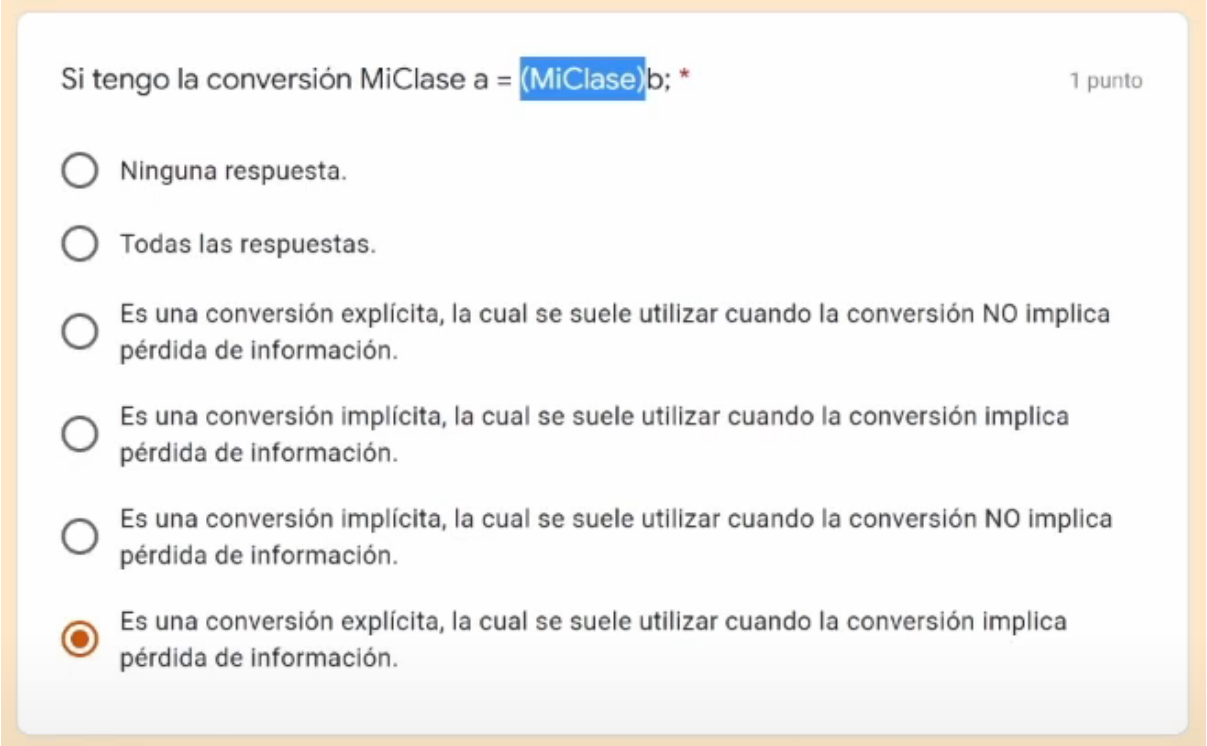
(Sobrecarga metodos)¿Cuáles de estas afirmaciones sobre la sobrecarga de métodos son correctas?



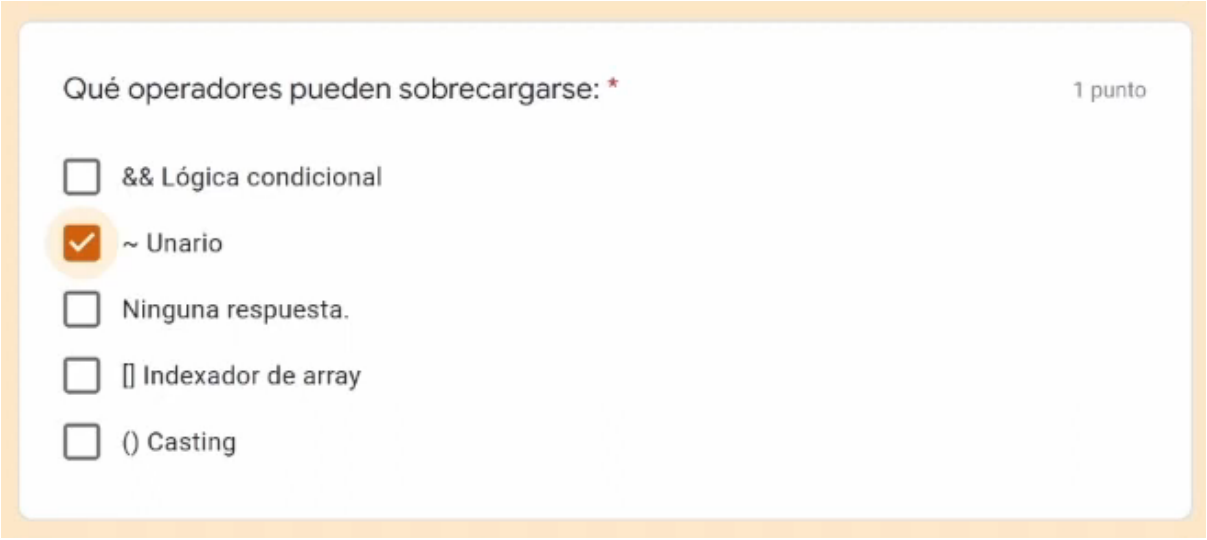
(Sobrecarga metodos)Un método se sobrecarga para:



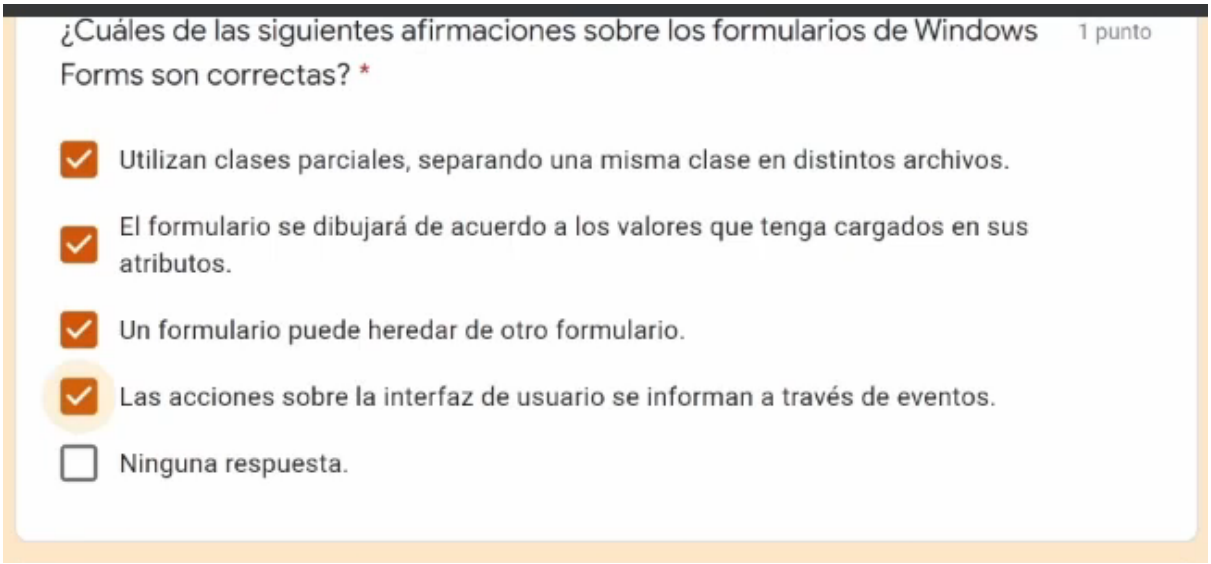
(Conversiones) Si tengo la conversión MiClase a = (MiClase)b;



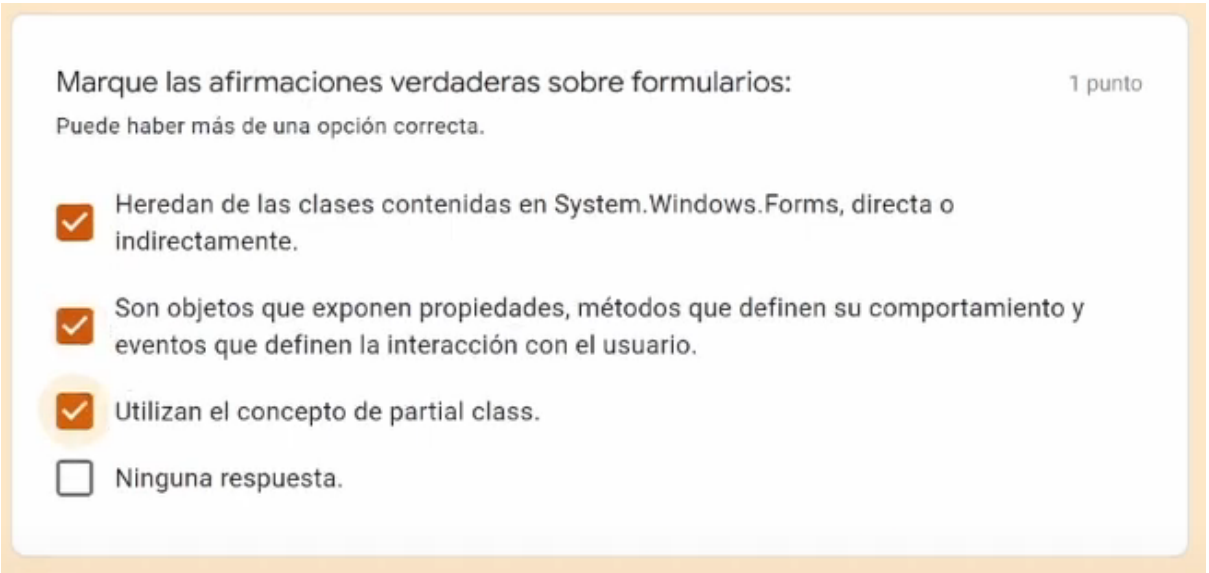
(Sobrecarga operadores)Qué operadores pueden sobrecargarse:



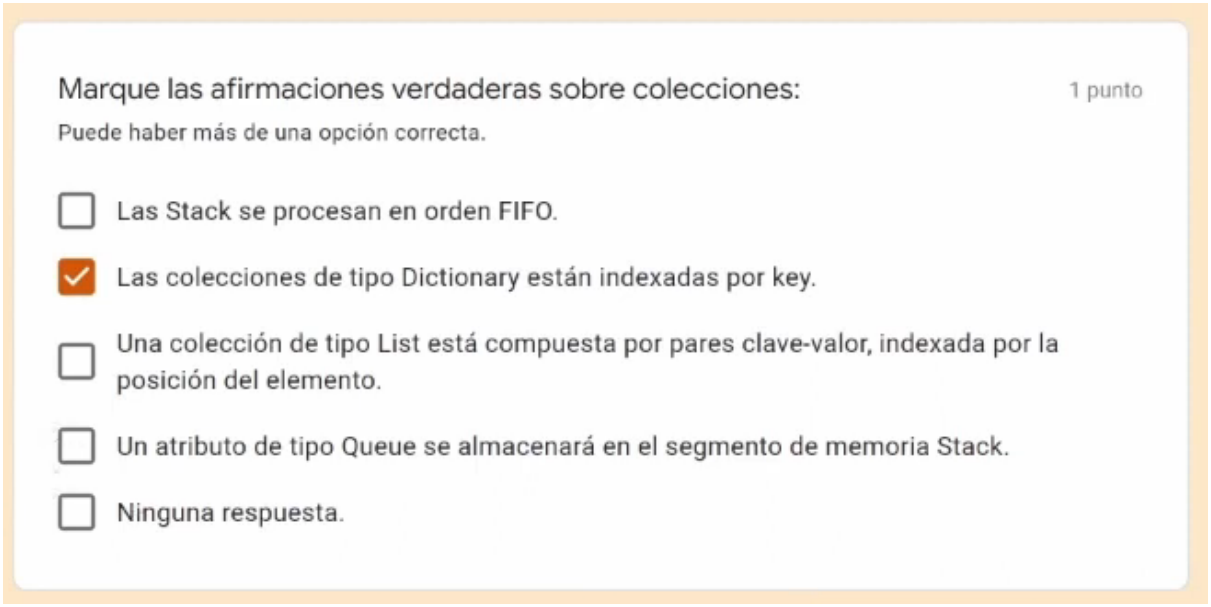
(Windows form)¿Cuáles afirmaciones sobre los formularios de Win Forms son correctas?



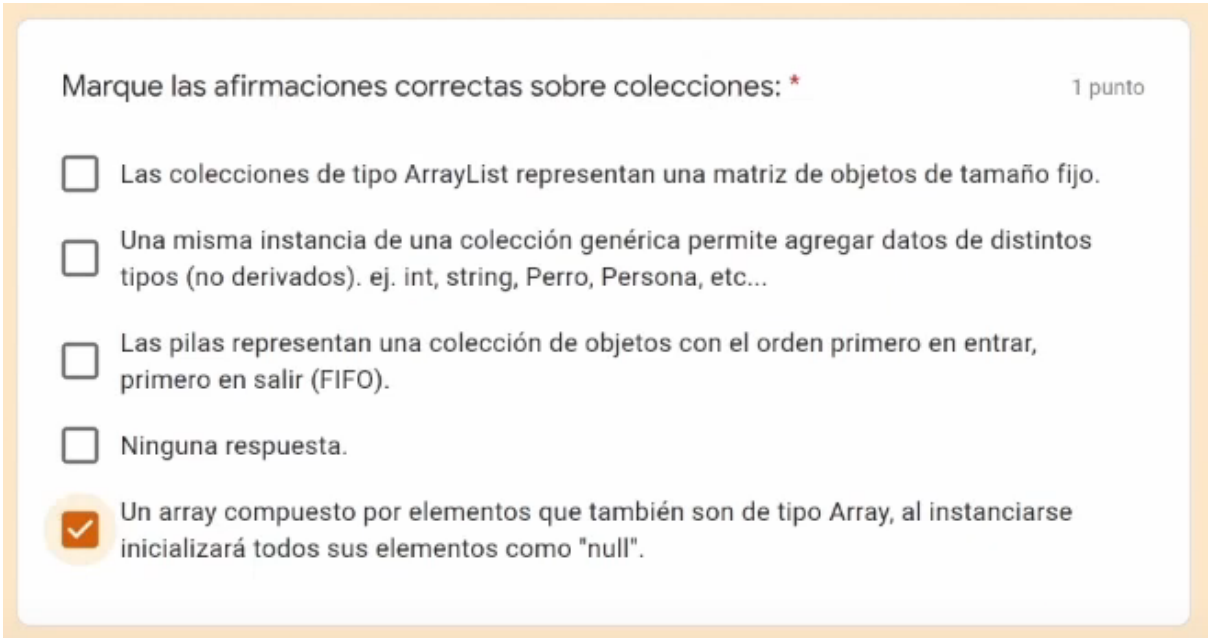
(Forms)Marque las afirmaciones verdaderas sobre formularios:



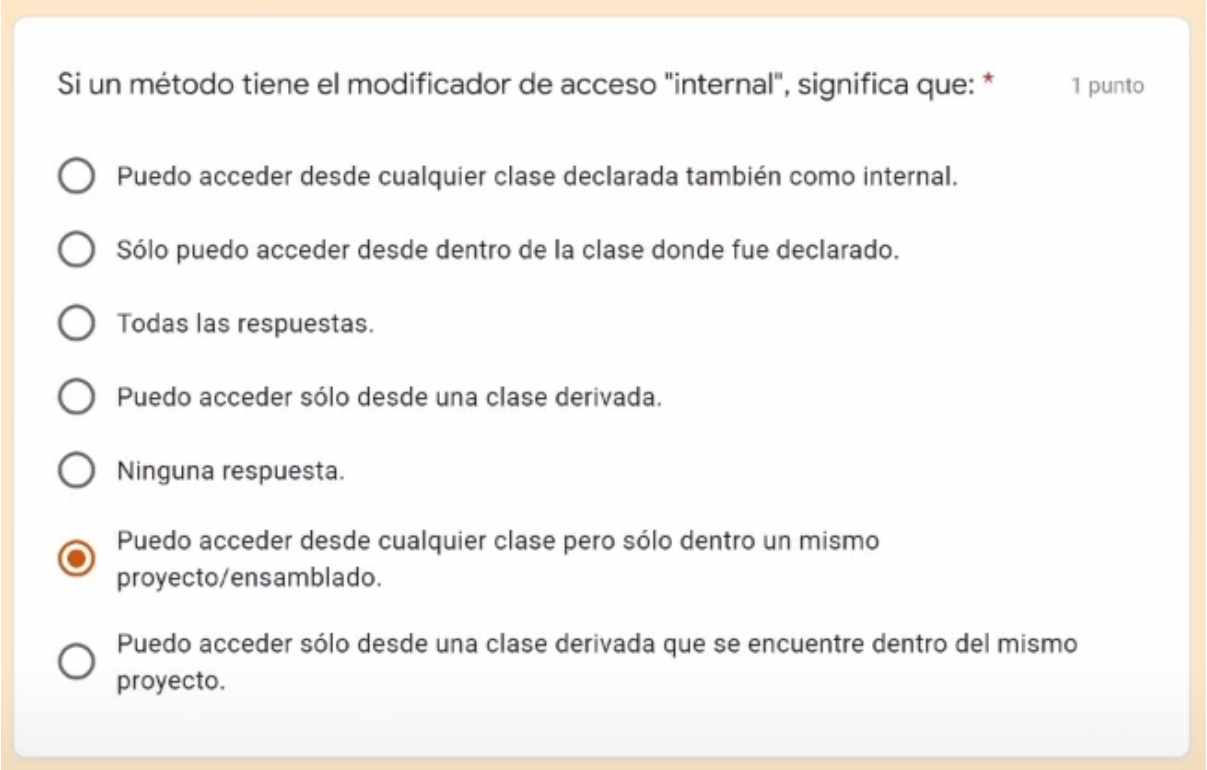
(Colecciones) Marque las afirmaciones verdaderas sobre colecciones:



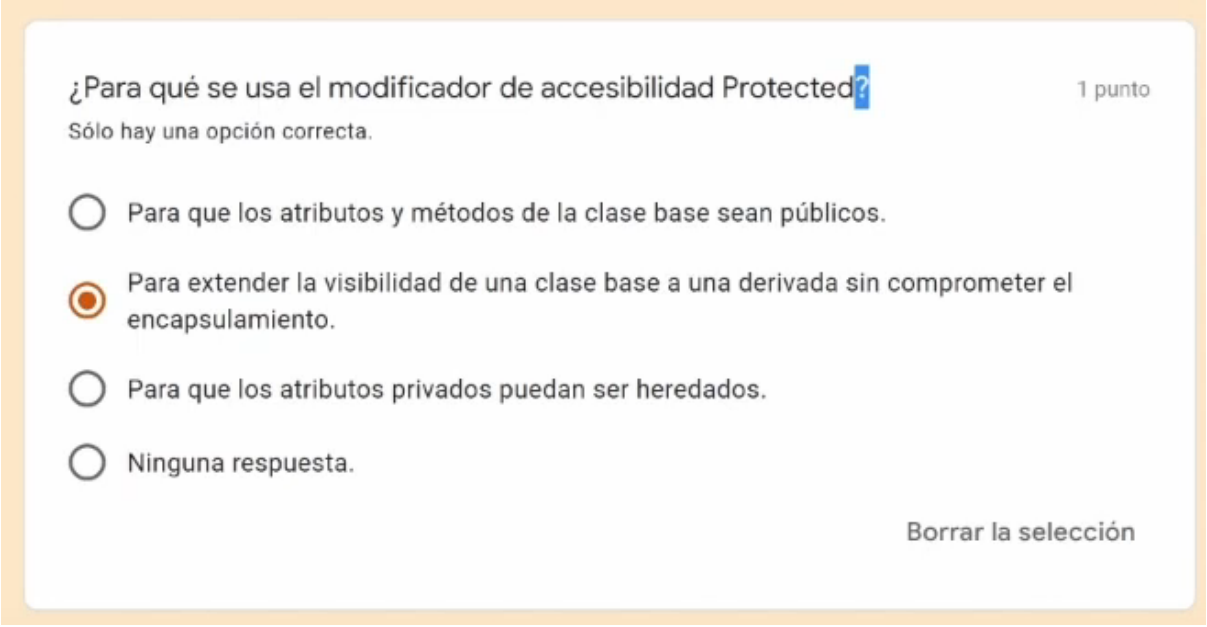
(Colecciones) Marque las afirmaciones verdaderas sobre colecciones:



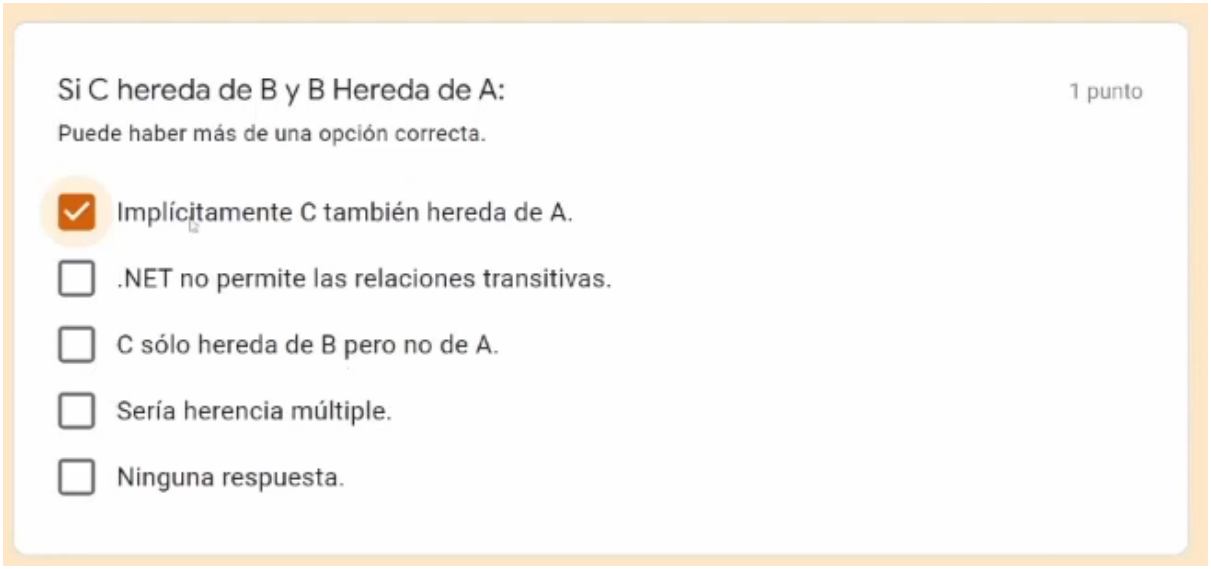
(Internal)Si un método tiene el modificador de acceso "internal", significa que:



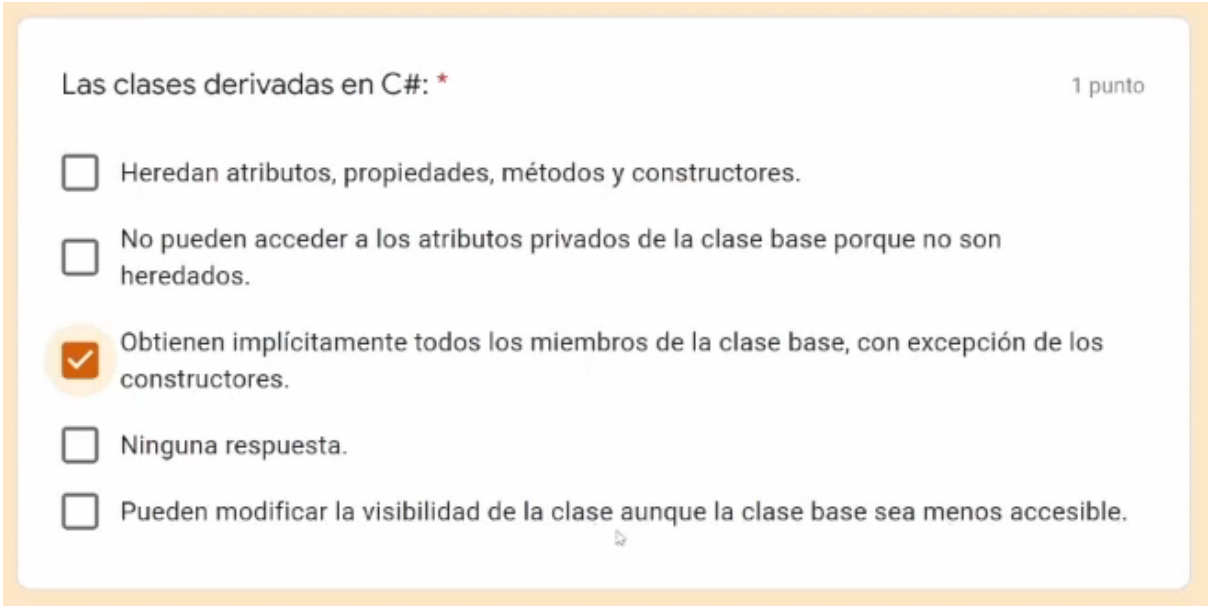
(Protected)¿Para qué se usa el modificador de accesibilidad Protected?



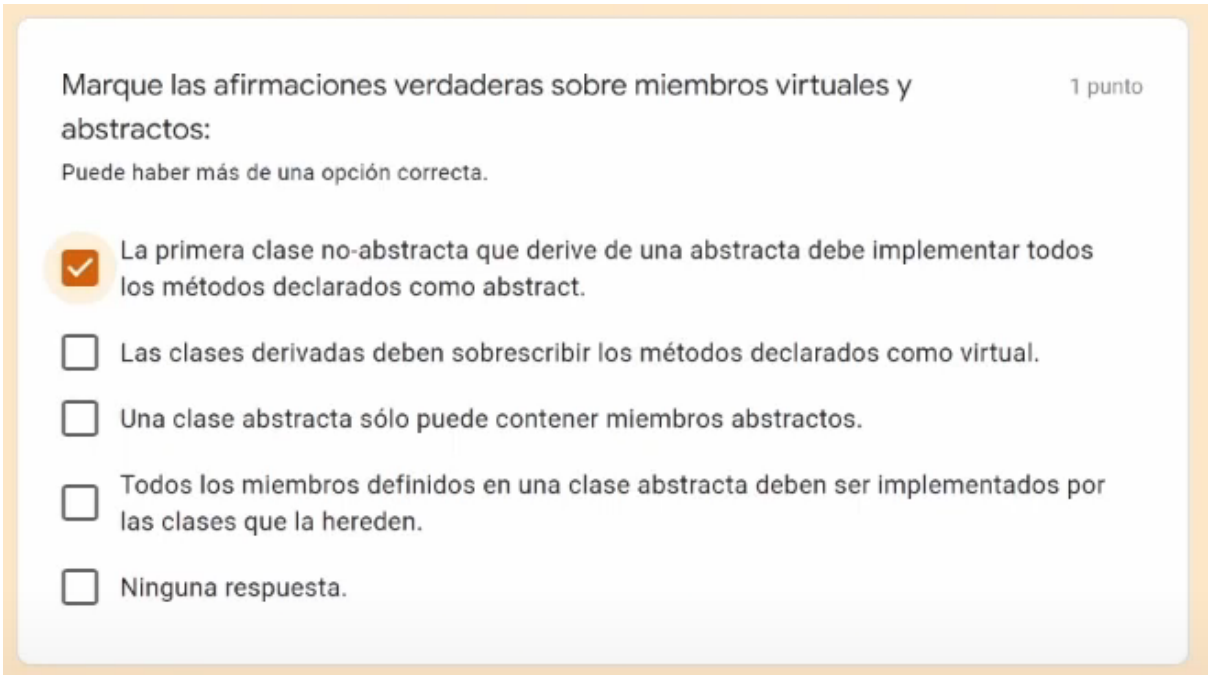
(Herencia)Si C hereda de B y B Hereda de A:



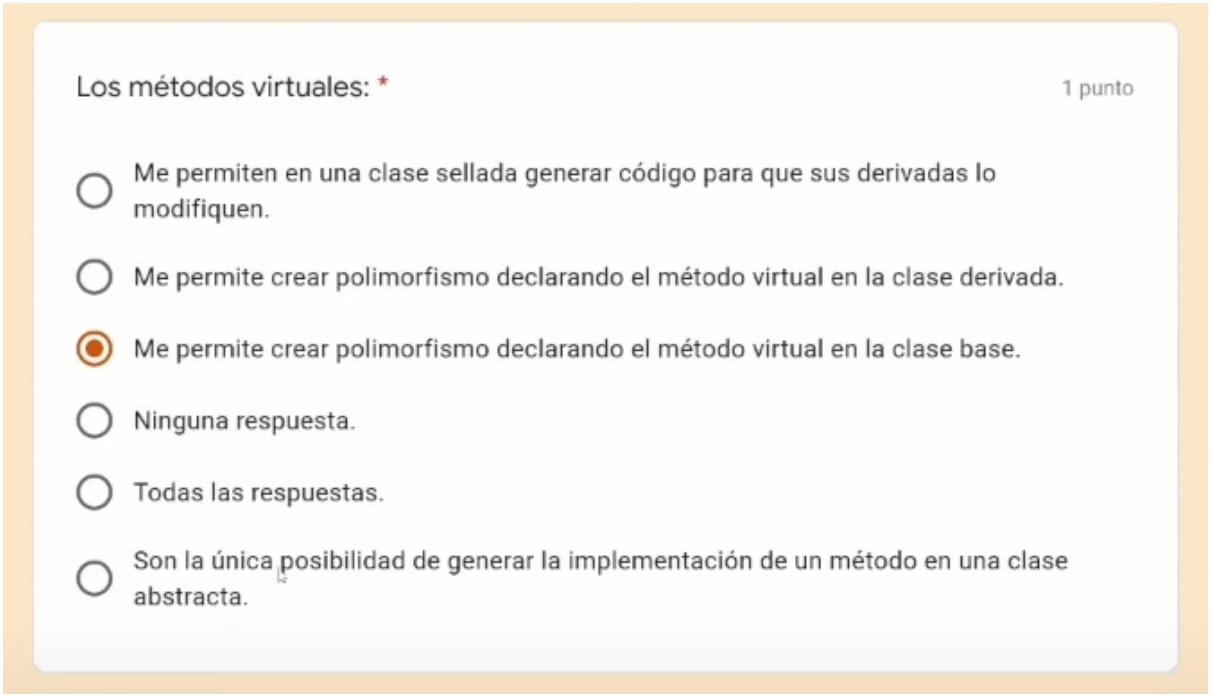
(Clases derivadas hijas)Las clases derivadas en C#:



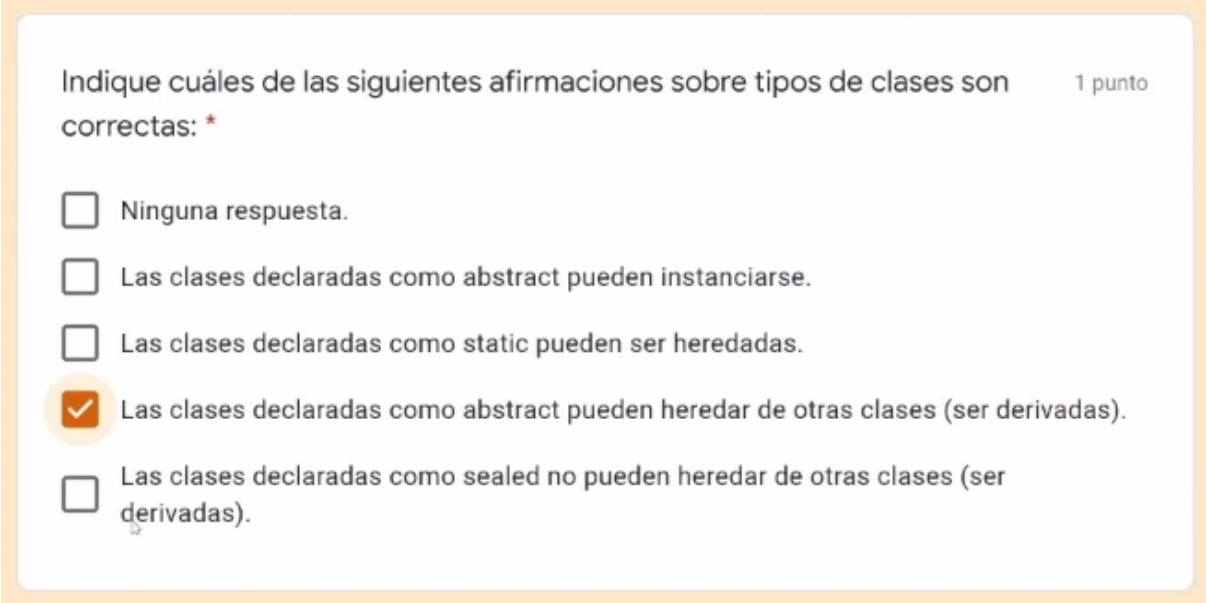
(virtuales y abstractas)Marque las afirmaciones verdaderas sobre miembros virtuales y abstractos:



Los métodos virtuales:



(clases)Indique cuáles de las siguientes afirmaciones sobre tipos de clases son correctas:



En base al código en la imagen marque las afirmaciones verdaderas.

En base al código en la imagen marque las afirmaciones verdaderas. 1 punto

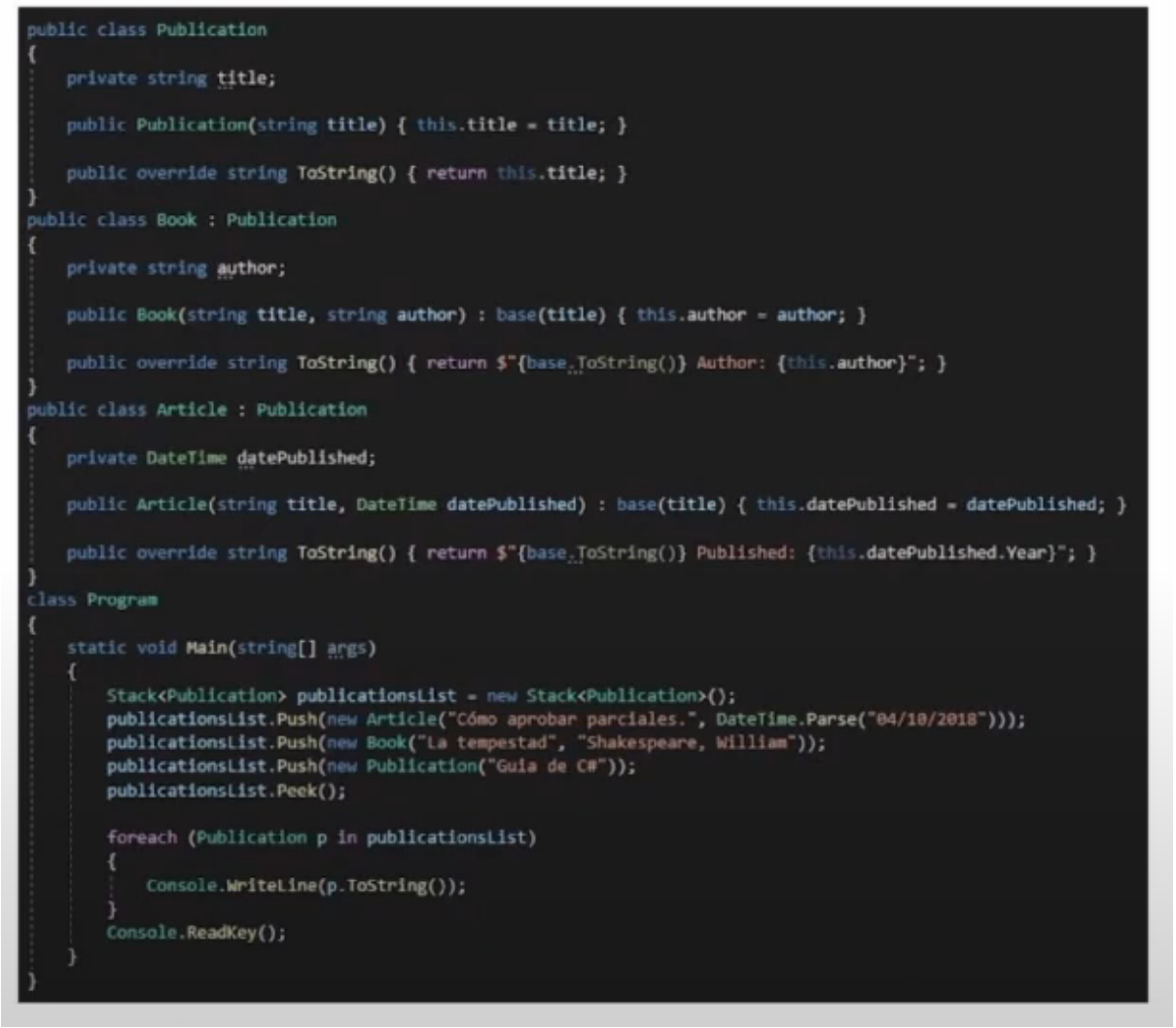
Puede haber más de una opción correcta.

```
abstract class A
{
    public abstract string MetodoLoco();
}
class B : A
{
    public override string MetodoLoco() { return "!false"; }
}
class C : B
{
    public override string MetodoLoco() { return $"{base.MetodoLoco()} its funny because its true" ; }
}
class Program
{
    static void Main(string[] args)
    {
        B b = new C();
        A a = b;
        Console.WriteLine( a.MetodoLoco() );

        Console.ReadKey();
    }
}
```

- [x] El tipo de la instancia en tiempo de ejecución determina la implementación del método a invocar.
- [ ] El tipo de la referencia en tiempo de compilación determina la implementación del método a invocar.
- [ ] La salida por consola es: "!false".
- [x] La salida por consola es: "!false its funny because its true".
- [ ] Error en tiempo de ejecución.
- [ ] Error en tiempo de compilación.

¿Cuál es la salida del programa de la imagen?

¿Cuál es la salida del programa de la imagen? 0 puntos



```
public class Publication
{
    private string title;

    public Publication(string title) { this.title = title; }

    public override string ToString() { return this.title; }
}
public class Book : Publication
{
    private string author;

    public Book(string title, string author) : base(title) { this.author = author; }

    public override string ToString() { return $"{base.ToString()} Author: {this.author}"; }
}
public class Article : Publication
{
    private DateTime datePublished;

    public Article(string title, DateTime datePublished) : base(title) { this.datePublished = datePublished; }

    public override string ToString() { return $"{base.ToString()} Published: {this.datePublished.Year}"; }
}
class Program
{
    static void Main(string[] args)
    {
        Stack<Publication> publicationsList = new Stack<Publication>();
        publicationsList.Push(new Article("Cómo aprobar parciales.", DateTime.Parse("04/10/2018")));
        publicationsList.Push(new Book("La tempestad", "Shakespeare, William"));
        publicationsList.Push(new Publication("Guia de C#"));
        publicationsList.Peek();

        foreach (Publication p in publicationsList)
        {
            Console.WriteLine(p.ToString());
        }
        Console.ReadKey();
    }
}
```