[Programación II] Auto-Evaluación Segundo Parcial

Puntos totales 17/18

El presente material es sólo una ayuda para auto evaluarse.

Resolverlo no es suficiente para garantizar la aprobación del parcial, se debe articular con las clases y el resto del material.

En las preguntas no se está evaluando el temario completo, sólo una fracción del mismo.

La forma de calificar estos formularios no está necesariamente relacionada a la forma de calificar los parciales.

Recomendaciones

* Responda la auto-evaluación una vez que haya estudiado para el parcial y no antes.
* Responda el cuestionario sin consultar apuntes, material teórico o internet.
* Lea atentamente los enunciados y las respuestas antes de responder.
* Una vez que terminó la auto-evaluación, repase los temas en los que le fue mal.

✓ La serialización binaria me servirá para: 1/1

- [x] Guardar en formato de bytes distintos tipos de datos. ✓
- [ ] Generar archivos sólo con la extensión .bin
- [ ] Ninguna respuesta.
- [ ] Serializar clases.
- [x] Serializar objetos. ✓

Comentarios

Se serializan instancias, con un estado específico (datos). Las clases como tales son moldes para crear instancias, no contienen datos.

La serialización en formato binario consiste en convertir la información a bytes.

La serialización se puede o no volcar a un archivo, no es obligatorio. Y si se hace, la extensión no define nada en particular, puede tener cualquiera. Las extensiones de ficheros son meramente para informar al sistema operativo del contenido del archivo.

✓ Teniendo en cuenta la siguiente línea de código, indicar las afirmaciones correctas: 1/1

```
System.IO.StreamWriter file = new System.IO.StreamWriter(archivo, true);
```

- [ ] Si el archivo existe, lo sobrescribirá.
- [x] Si declaro e instancio el objeto StreamWriter en una sentencia "using", no deberé preocuparme por cerrar el vínculo con el archivo. ✓
- [ ] Ninguna respuesta.
- [x] Si el archivo existe, agregará información al mismo. ✓
- [x] Al finalizar, deberé cerrar el vínculo con el archivo para que no quede tomado. ✓
- [ ] Me servirá para serializar en formato binario.

Comentarios

StreamWriter es una clase que permite escribir en un archivo. No serializa, sólo transmite y escribe la información en el archivo.

Es muy importante cerrar el vínculo con el archivo una vez realizados los cambios (si llegó a ser abierto), de otra forma quedará "tomado".

El segundo argumento del constructor inicializa la propiedad append, la misma indica si se anexará el nuevo texto (true) o se sobrescribe el contenido (false).

La sentencia using se encarga de cerrar la transmisión y liberar los recursos automáticamente, por lo que no tendremos que preocuparnos por cerrarla. Internamente lo que hace es llamar al método Dispose() cuando el flujo de ejecución abandona el scope.

✓ Al ejecutar un thread: 1/1

- [ ] Finalizará si se culmina la ejecución del hilo principal.
- [x] Cuando finalice liberará el bloque de memoria que ocupaba. ✓
- [ ] Proporcionaremos un medio apropiado para que los objetos puedan señalizar cambios de estado que pueden resultar útiles para los usuarios/clientes de ese objeto.
- [ ] Podremos ejecutar el método Abort más de una vez para el mismo método sin errores.
- [ ] Ninguna respuesta.

Comentarios

Entendemos como hilo principal al primero que se ejecuta. Los hilos son independientes, incluso del principal. Que el hilo principal finalice no implica que finalicen los otros hilos del proceso.

Los hilos finalizados no pueden volver a iniciarse, hay que generar una nueva instancia. Por lo tanto no es posible utilizar el método Abort más de una vez sin errores.

Señalizar cambios de estado no tiene que ver con hilos, en todo caso está más relacionado a eventos.

✓ Suponiendo que el siguiente código se encuentra dentro de un proyecto de Consola, teniendo las referencias a las librerías necesarias implementadas correctamente, contestar que hará el siguiente código según las opciones dadas: 1/1

```
class Prueba
{
    public Prueba(string mensaje)
    {
        Thread thread = new Thread(new ParameterizedThreadStart(Imprime));

        thread.Start(mensaje);
    }

    void Imprime(object o)
    {
        Console.WriteLine((string)o);
    }
}
```

- [ ] Error en tiempo de ejecución.
- [ ] Imprime en el hilo principal un mensaje por consola.
- [ ] Ninguna respuesta.
- [ ] Error en tiempo de diseño.
- [x] Imprime en un hilo secundario un mensaje por consola. ✓

Comentarios

Entendemos como hilo principal al primero que se ejecuta.

El método Imprime se ejecuta en un hilo secundario mostrando un mensaje por consola.

✓ Indique las afirmaciones correctas sobre tipos genéricos: 1/1

- [ ] Los métodos pueden recibir tipos genéricos, pero no retornarlos.
- [x] Se pueden utilizar en clases, métodos, atributos y propiedades. ✓
- [ ] Ninguna respuesta.
- [ ] Las interfaces no pueden especificar tipos genéricos.
- [ ] Sólo se puede tener un tipo genérico por clase.
- [x] Al generar la clase, reemplazaré por un comodín lo que podría ser un tipo de dato específico. ✓

Comentarios

Los tipos genéricos nos permiten trabajar con menos restricciones sin dejar de contar con seguridad de tipos.

Consiste en utilizar un comodín (tipo genérico) que será reemplazado por un tipo concreto durante la compilación.

Tantos clases como interfaces y métodos pueden utilizar tipos genéricos. Los métodos pueden tenerlos como parámetro de entrada o tipo de retorno. También los tipos de atributos y propiedades pueden ser genéricos.

En el siguiente código hay 2 errores, encontrarlos y corregirlos.

```
interface IPrueba<Tu>
{
    private void MetodoInterfaz();
    void MetodoLanzaException(Tu aux);
}
public class MiClase : IPrueba<string>
{
    public void MetodoInterfaz()
    {
        string rta = this.MetodoLanzaException("Error!");
    }

    public void MetodoLanzaException(string aux)
    {
        throw new Exception(aux);
    }
}
```

1 - MetodoInterfaz tiene un modificador de acceso en la interfaz, no deberia tenerlo.
2 - MEtodoLanzarExcepcion solo retorna void, no se puede asignar a una variable.

Comentarios

** No se pueden especificar métodos privados en una interfaz.*

** MetodoLanzaExcepcion no retorna un string ni nada, no se puede asignar a una variable.*

✗ Indique cuáles de las siguientes afirmaciones sobre hilos son correctas: 0/1

- [ ] Ninguna respuesta.
- [x] Mediante estos, una tarea que puede ser ejecutada al mismo tiempo que otra tarea. ✓
- [ ] En el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y los recursos son liberados.
- [x] Si quisiéramos lanzar un hilo parametrizado, deberíamos utilizar el delegado ParameterizedThreadStart. ✓
- [x] Si deseáramos detener un hilo en ejecución, utilizaríamos el método Abort. ✓

Respuesta correcta

- [x] Mediante estos, una tarea que puede ser ejecutada al mismo tiempo que otra tarea.
- [x] En el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y los recursos son liberados.
- [x] Si quisiéramos lanzar un hilo parametrizado, deberíamos utilizar el delegado ParameterizedThreadStart.
- [x] Si deseáramos detener un hilo en ejecución, utilizaríamos el método Abort.

En el contexto de una competencia de programación sin IDEs, Victoria encontró 3 errores en el código que el resto de su equipo pasó por alto. Amelia, una de sus compañeras, señaló que cometieron una mala práctica. ¿Podrás encontrar los errores y corregirlos como hicieron Victoria y Amelia?

```
interface IExponer<S>
{
    S Datos { get; }
}

class ClaseBase<V>
{
    public V entero;
    public ClaseBase(long i)
    {
        this.entero = i;
    }
}

class ClaseEjecutor<int> : ClaseBase<T>, IExponer<T>
{
    public ClaseEjecutor(T e)
        : base(e)
    {
    }

    public T Datos
    {
        get
        {
            return base.entero;
        }
    }
}

static void Main(string[] args)
{
    ClaseEjecutor<string> ejecutor = new ClaseEjecutor("Hola");
    Console.WriteLine(ejecutor.Datos);
}
```

1 - La ClaseBase no deberia tener atributos publicos.
2 - el constructor de ClaseBase deberia recibir V y no long.
3 - ClaseEjecutor en el main deberia ser instanciada con un <string> en la firma despues del new.

Comentarios

Errores encontrados por Victoria:

** El constructor de ClaseBase debería recibir el tipo genérico V en vez de long.*

** En la declaración de la clase genérica ClaseEjecutor se debería reemplazar <int> por <T>*

** Cuando se instancia ClaseEjecutor en el método Main, se debe agregar <string> seguido del identificador de la clase.*
Quedaría:
ClaseEjecutor<string> ejecutor = new ClaseEjecutor<string>("Hola");

Errores encontrados por Amelia:

** Es una buena práctica que los atributos no tengan visibilidad pública. Por lo tanto, el atributo "entero" de ClaseBase debería ser private.*

✓ ¿Cuáles de las siguientes estructuras son válidas? 1/1

LOS COMENTARIOS TAMBIÉN DEBEN SER CORRECTOS

```
// OPCION A
try { ... }

// OPCION B
try { /* aqui se lanza la excepción */ }
catch (Exception e) { /* aquí se controla la excepción */ }

// OPCION C
try { /* aquí se controla la excepción */ }
catch (Exception e) { /* aquí se lanza la excepción */ }

// OPCION D
try { ... }
catch (Exception e) { ... }
finally { /* se ejecutará si no se lanzó ninguna excepción  */ }

// OPCION E
try { ... }
finally { ... }

// OPCION F
try { ... }
catch (Exception e) { ... }
finally { /* se ejecutará siempre  */ }
```

- [ ] Opción A
- [x] Opción B ✓
- [ ] Opción C
- [ ] Opción D
- [x] Opción E ✓
- [x] Opción F ✓
- [ ] Ninguna respuesta.

Comentarios

A) Incorrecto. Un bloque Try sólo no existe, siempre debe ir acompañado de un catch o un finally. No tiene sentido usar un try si no habrá un bloque que determine qué hacer en caso de una excepción.

B) Es correcto. El bloque try evalúa si se lanzó una excepción y el catch contiene el código para controlarla.

C) Incorrecto. La excepción se controla en el bloque catch, no en el try.

D) Es correcto. El bloque finally se ejecuta SIEMPRE, se haya lanzado o no una excepción.

E) Incorrecto. La estructura try-finally es válida en C#.

F) Es correcto. El bloque finally se ejecutará SIEMPRE, haya ocurrido o no una excepción.

✓ Al generar un método de extensión: 1/1

- [ ] Creo una nueva clase que hereda de una clase preexistente.
- [x] Agrego funcionalidades a una clase preexistente. ✓
- [ ] Ninguna respuesta.
- [ ] Hago referencia a la clase extendida mediante la palabra reservada base.
- [ ] Anulo los métodos de la clase extendida.

Comentarios

Los métodos de extensión nos permiten agregar funcionalidad a tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original.

✓ Cuando accedemos a una base de datos utilizando la API ADO.NET: 1/1

- [x] SqlConnection administrará la conexión con un servidor. ✓
- [x] El objeto SqlCommand se conectará a la base de datos por medio de otro objeto. ✓
- [ ] Ninguna respuesta.
- [x] Podremos utilizar el mismo SqlCommand para ejecutar consultas en distintos servidores, sólo cambiando la conexión. ✓
- [x] Se pueden ejecutar sentencias al menos de dos formas: ExecuteNonQuery y ExecuteReader. ✓

Comentarios

Todas son correctas.

SqlConnection es la clase encargada de administrar la conexión con la base de datos, mientras que SqlCommand se encarga de realizar consultas y ejecutar comandos.

✓ ¿Qué son los test unitarios? 1/1

- [ ] Son una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software.
- [x] Son pruebas para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto. ✓
- [ ] Son aquellos que prueban que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo.
- [ ] Ninguna respuesta.

Comentarios

Los tests unitarios sirven para escribir casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto.

Los tests funcionales o end-to-end son una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software.

Los tests integrales son aquellos que prueban que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo.

✓ Al ejecutar las siguientes líneas de código dentro de una clase del tipo Form, sucederá que: 1/1

MiMetodo NO contiene un bucle infinito.

```
Thread t = new Thread(MiMetodo);
t.Start();
```

- [x] Se podrá cancelar el hilo a través del método Abort. ✓
- [x] La propiedad IsAlive me retornará el estado de ejecución del hilo. ✓
- [x] Se lanzará un nuevo hilo, totalmente independiente del actual. ✓
- [ ] Ninguna respuesta.
- [ ] Se ejecutará el método MiMetodo hasta que se cancele el hilo por código.
- [x] Si se quiere cancelar un thread inactivo, lanzará una excepción. ✓

Comentarios

Los nuevos hilos son INDEPENDIENTES del hilo donde fueron creados.

Un hilo vivirá / se ejecutará hasta que finalice la ejecución del método que invoca o se aborte a través del método Abort, lo que ocurra primero.

La propiedad IsAlive retorna "true" si el hilo se encuentra en ejecución, "false" de lo contrario.

✓ ¿Cómo hago una consulta en ANSI SQL a una tabla provincia filtrando por id igual a 2? 1/1

- [ ] SELECT ALL FROM provincia WHERE id = 2;
- [ ] SELECT 2 FROM provincia;
- [ ] GET ALL FROM provincia PROVINCIA WHERE id = 2;
- [ ] Provincia.getAll(2);
- [x] SELECT * FROM provincia WHERE id = 2; ✓
- [ ] Ninguna respuesta.
- [ ] SELECT * provincia WHERE id = 2;

Comentarios

La sintaxis correcta es la señalada, todas las demás contienen errores.

✓ Cuando realice un test unitario: 1/1

Utilizando Visual Studio Unit Testing Framework (MSTest)

- [x] Cada método llevará la etiqueta [TestMethod] ✓
- [ ] Nuestra clase test heredará de UnitTest.
- [x] A través de métodos estáticos de la clase Assert podré verificar el resultado de cada test. ✓
- [x] Podré verificar cómo se comporta una porción del código interpretando valores de variables, excepciones lanzadas, etc. ✓
- [ ] Ninguna respuesta.

Comentarios

El framework de pruebas MSTest nos permite automatizar pruebas unitarias para verificar el comportamiento de unidades de código concretas frente a distintos casos de pruebas.

Los métodos estáticos de la clase Assert nos permitirán verificar que el resultado sea el esperado, caso contrario lanzarán una excepción para informarnos qué la prueba ha fallado.

Cada test unitario está contenido en un método decorado con la etiqueta [TestMethod].

Las clases de test unitarios no heredan necesariamente de ninguna otra, pero tienen el decorador [TestClass].

✓ Si tengo el siguiente código. ¿Qué afirmaciones PODRÍAN ser correctas? 1/1

```
public class MiClase<T, S> : A<T>, B<S>, C<T, S>
    where T : MiTipo {}
```

- [ ] A y B son interfaces, C es una clase.
- [ ] T y S deberán ser de tipo "MiTipo"
- [x] A, B y C son interfaces. ✓
- [ ] A y B son clases.
- [x] A es una clase, B y C interfaces. ✓
- [ ] Ninguna respuesta.
- [x] T deberá ser de tipo "MiTipo" o de algún tipo que herede o implemente este tipo de dato. ✓

Comentarios

C# no admite herencia múltiple. Por lo tanto, sólo se puede heredar de una clase.

Si se hereda y al mismo tiempo se implementa una interfaz, la herencia siempre se indica primero.

En este código, sólo el tipo genérico "T" tiene una restricción para ser o heredar de "MiTipo". El tipo "S" no tiene restricciones.

✓ Indique cuáles de las siguientes afirmaciones sobre métodos de extensión son correctas: 1/1

- [x] Su utilización será mediante una instancia de la clase extendida. ✓
- [ ] Se utilizarán mediante el identificador de la clase extendida.
- [x] Permiten adicionar métodos a tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original. ✓
- [x] El método y la clase donde es declarado deberán ser estáticos. ✓
- [x] Permiten extender clases selladas. ✓
- [ ] Ninguna respuesta.

Comentarios

Los métodos de extensión nos permiten adicionar métodos a tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original. Esto incluye clases del sistema o con código privativo.

Deben estar dentro de una clase estática y, si bien se declaran como un método estático, se invocan como si fueran un método de instancia de la clase extendida.

Las clases selladas, las cuales no se pueden extender mediante herencia, pueden extenderse mediante este mecanismo.

✓ Indique cuáles de las siguientes afirmaciones sobre serialización son correctas: 1/1

- [ ] Podemos serializar en archivos de texto plano.
- [ ] Sólo se pueden serializar en formato binario atributos y propiedades públicas.
- [x] Los objetos que se deseen serializar en formato XML deben tener un constructor público sin parámetros. ✓
- [ ] Ninguna respuesta.
- [ ] Los objetos que se deseen serializar en formato binario no necesitan ningún agregado o característica en particular.

Comentarios

El texto plano es un texto sin formato, y el proceso de serializar implica implícitamente aplicar un formato a los datos.

En C#, cuando serializamos en formato binario se pueden serializar datos almacenados tantos en atributos privados como públicos. Los tipos a serializar en formato binario deben marcarse con la etiqueta [Serializable] y deben tener un constructor público sin parámetros.

En c#, los tipos a serializar en formato XML deben tener un constructor público sin parámetros y sólo se podrán serializar atributos y propiedades públicas.

✓ Indique cuáles de las siguientes afirmaciones sobre interfaces son correctas: 1/1

- [ ] Al realizar la herencia de una clase abstracta que implementa una interfaz, podré implementar los métodos directamente en la clase que la hereda, sin deber implementarlos en la clase base.
- [ ] Los métodos se definirán con la palabra reservada abstract ya que no contienen código.
- [x] La clase que las implemente deberá darle una implementación a cada una de las operaciones especificadas por la interfaz. ✓
- [x] Son un contrato que establece una clase en el cual la misma asegura que implementará un conjunto de operaciones. ✓
- [ ] Ninguna respuesta.
- [ ] Contienen métodos, atributos y propiedades.
- [x] Todos los elementos que las componen son públicos. ✓

Comentarios

Las interfaces son un contrato que establece una clase en el cual la misma asegura que implementará un conjunto de operaciones.

La clase que implementa la interfaz debe darle una implementación a todas las operaciones (métodos) definidas por la misma. Las clases abstractas no son una excepción.

Las interfaces no tienen estado, es decir, no especifican atributos. Sí se pueden especificar propiedades.

Es parte de la naturaleza de las interfaces que todos sus elementos sean públicos, ya que especifican una serie de servicios que una clase debe exponer.

Las especificaciones contienen únicamente la firma de los métodos, sin implementación (código). Similar a los métodos abstractos, pero por tratarse de una característica general de las interfaces no es necesario ni correcto utilizar la palabra abstract.

✓ Indique cuáles de las siguientes afirmaciones sobre eventos son correctas: 1/1

- [x] Los eventos permiten a una clase u objeto enviar notificaciones a otras clases u objetos cuando sucede algo en particular. ✓
- [x] Para asociar un manejador a un evento se usa el operador +=. ✓
- [ ] Sólo puede haber un manejador o subscriptor por cada evento.
- [ ] Cada manejador o subscriptor puede estar asociado a un sólo evento.
- [ ] Es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto.
- [x] Su definición depende de un delegado. ✓
- [ ] Ninguna respuesta.

Comentarios

Los delegados son tipos que permiten encapsular una referencia a un método con una firma específica. Con firma nos referimos a tipo de retorno y parámetros de entrada. Podremos invocar, a través del delegado, al método al que apunta.

Un evento es una forma más de comunicación entre objetos que permite enviar una notificación cuando ocurre algo en particular (un cambio de estado, una acción de un usuario en una interfaz gráfica, etc).

Un manejador de un evento es un método (encapsulado en un tipo delegado) que se ejecutará cuando se lance un evento. Para esto hay que asociar el manejador al evento con el operador "+=".

A un evento pueden subscribirse múltiples manejadores y un manejador puede estar asociado a uno o más eventos.
Si no hay manejadores asociados el evento no se ejecutará (lanzará una excepción si no se controla).

Un evento es de un tipo delegado. Esto le permite, con seguridad de tipos, almacenar la referencia a los manejadores e invocarlos cuando se lanza el evento.