Universidad Tecnológica Nacional
Facultad Regional Avellaneda

Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos

Materia: Programación II

Apellido:		Fecha:	
Nombre:		Docente(2):	
División:		Nota(2):	
Legajo:		Firma(2):	

Instancia(1):	PP		RPP		SP	X	RSP		FIN	

(1) Las instancias validas son: 1er Parcial (**PP**), Recuperatorio 1er Parcial (**RPP**), 2do Parcial (**SP**), Recuperatorio 2do Parcial (**RSP**), Final (**FIN**). Marque con una cruz.
(2) Campos a ser completados por el docente.

Marcar la/s opciones correctas para cada punto. Justificar si lo cree necesario:

1. Excepciones:
   a. Son errores del sistema que cierra nuestro programa sin darnos más opción.
   b. Cuando se detiene el flujo actual del programa, y si no se hace nada, el programa dejará de funcionar.
   c. Hay dos tipos: generadas por un programa en ejecución y las generadas por Common Language Runtime (CLR).
   d. Se controlan por medio de estructuras repetitivas.
   e. Sólo las puede lanzar el sistema operativo.
   f. Una vez lanzada, no se puede volver a lanzar.
   g. Todas las anteriores.

2. Cuáles de las siguientes estructuras son válidas para controlar una excepción que fue lanzada (lea los comentarios dentro de cada bloque, estos también deben ser correctos):
   a. **try** { ... }
   b. **try** { // aquí se lanza la excepción } **catch** (Exception e) { // aquí se controla la excepción }
   c. **try** { // aquí se controla la excepción } **catch** (Exception e) { // aquí se lanza la excepción }
   d. **try** { ... } **catch** (Exception e) { ... } **finally** { // se ejecutará si no se lanzó ninguna excepción }
   e. **try** { ... } finally { ... }
   f. **try** { ... } **catch** (Exception e) { ... } **finally** { // se ejecutará siempre }
   g. Todas las anteriores.
   h. Ninguna de las anteriores.

3. Los Test Unitarios:
   a. Son una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software.
   b. 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.
   c. Son aquellos que prueban que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo.
   d. Todas las anteriores.
   e. Ninguna de las anteriores.

4. Las interfaces explícitas nos permiten definir:
   a. Sólo métodos.
   b. Sólo atributos.
   c. Sólo propiedades.
   d. Métodos, propiedades y atributos.
   e. Métodos y propiedades.
   f. Métodos y atributos.
   g. Atributos y propiedades.

5. Si tengo *class T : J, B {}*
    a. T es una clase, J es una clase y B una interfaz.
    b. T es una clase, J es una clase al igual que B.
    c. T es una clase, J es una interfaz al igual que B.
    d. El fragmento de código es erróneo.
    e. Ninguna de las anteriores.

6. Si tengo class MiClase : IDatos<T> {}
    a. T es un tipo de dato genérico.
    b. T es del tipo MiClase.
    c. T es del tipo IDatos.
    d. El fragmento de código es erróneo.
    e. Todas las anteriores.
    f. Ninguna de las anteriores.

7. Archivos:
    a. En qué archivos podemos serializar:
        Rta:
    b. Para pasar una clase completa a archivo binario, ¿qué marca debemos colocarle previo a su declaración class MiClase?
        Rta:
    c. ¿Qué atributos podemos serializar mediante archivos XML?
        Rta:
    d. ¿Qué atributos podemos serializar mediante archivos de texto?
        Rta:

8. Threads:
    a. Mediante estos, una tarea que puede ser ejecutada al mismo tiempo que otra tarea.
    b. Es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo.
    c. En el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y los recursos son liberados.
    d. Todas las anteriores.
    e. Ninguna de las anteriores.

9. Eventos:
    a. Es el modo que tiene una clase en particular de proporcionar notificaciones a sus clientes cuando ocurre algo en particular dentro del objeto.
    b. Es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto.
    c. Proporcionan un medio apropiado para que los objetos puedan señalizar cambios de estado que pueden resultar útiles para los clientes de ese objeto.
    d. Se ejecutan automáticamente sin que tengamos que hacer nada.
    e. Contienen algo similar a una lista de punteros a funciones de C++.
    f. Se implementan mediante delegados.
    g. Todas las anteriores.

10. En los archivos de texto:
    a. Se puede agregar información en cualquier momento al archivo.
    b. Una vez creado un archivo, si volvemos a decirle que lo cree lanza una excepción el sistema.
    c. Para poder generar un Stream, debemos informar el tipo de dato a guardar.
    d. Utilizaremos BinaryFormatter.
    e. Todas las anteriores.
    f. Ninguna de las anteriores.

11. Las bases de datos:
    a. Deberán utilizar SqlCommand para ejecutar consultas.
    b. Son archivos XML con información organizada.
    c. Se cargan datos por medio de StreamWriter.
    d. Se deberá generar una conexión para poder acceder a los datos.
    e. Todas las anteriores.
    f. Ninguna de las anteriores.

12. Completar los comentarios del siguiente código describiendo que hará la línea siguiente a cada // o la consigna planteada en el mismo comentario:

```
    static bool seguir;
    static void Main(string[] args)
    {
        Program.seguir = true;
        //
        Thread t = new Thread(Program.ProbarThreads);
        //
        t.Start();

        //
        Thread.Sleep(2000);
        Program.seguir = false;

        Console.ReadKey();

        // ¿Qué imprime esta línea?:
        Console.WriteLine(t.IsAlive);
        Console.ReadKey();
    }

    /// <summary>
    /// ¿Qué hará ProbarThreads?
    /// </summary>
    static void ProbarThreads()
    {
        int i = 0;
        while (seguir)
        {
            Console.WriteLine(i);
            i++;
        }
    }
```

13. Marque los errores del siguiente código mediante un círculo y reescribalo correctamente:

```
class MiOtraClase { }
interface IAlgo
{
    public string MiMetodo(int dato);
}
class MiClase : IAlgo, MiOtraClase
{
    string MiMetodo(long dato)
    {
        string retorno = "";
        for (int i = 1; i <= dato; i = i + 2)
            retorno += String.Format("{0} ", dato);
        return retorno;
    }
}
```