Universidad Tecnológica Nacional
Facultad Regional Avellaneda

Técnico Universitario en Programación – Programación 02

Cuestionario Teórico

Contenido

[Módulo - 01] C# y .NET....2
[Módulo - 02] Programación orientada a objetos....2
[Módulo - 03] Miembros estáticos y de instancia: 2
[Módulo - 04] Namespaces y directivas: 3
[Módulo - 05] Objetos: 3
[Módulo - 06] Constantes: 4
[Módulo - 07] Constructores:....4
[Módulo - 08] Sobrecargas (Overload):....4
[Módulo - 09] Sobrecarga de métodos: 4
[Módulo - 10] Sobrecarga de constructores:....5
[Módulo - 11] Sobrecarga de operadores:....5
[Módulo - 12] Formularios:....6
[Módulo - 13] Arrays:....6
[Módulo - 14] Strings: 6
[Módulo - 15] Colecciones:....7
[Módulo - 16] Propiedades:....8
[Módulo - 17] Enumerados: 8
[Módulo - 18] Indexadores:....8
[Módulo - 19] Encapsulamiento:....9
[Módulo - 20] Herencia: 9
[Módulo - 21] Polimorfismo: 9
[Módulo - 22] Sobreescritura de métodos (Override): 10
[Módulo - 23] Clases y miembros abstractos:....10

[Módulo - 01] C# y .NET

(001) - ¿Qué es el **CLR**?

(002) - Describa el proceso de compilación de C#.

(003) - ¿Qué es el **CTS**?

(004) - ¿Qué es un tipo por referencia (**Reference Type**)? ¿En qué se diferencia de un tipo por valor (**Value Type**)?

(005) - Explique las diferencias entre variables **escalares** y **no escalares**.

(006) - ¿Cuál es el punto de entrada (**entry point**) para los programas en C#?

(007) - ¿Cuál es la diferencia entre una conversión **implícita** y una **explícita**?

[Módulo - 02] Programación orientada a objetos

(008) - ¿Qué propone el **paradigma orientado a objetos**? ¿Qué es un paradigma?

(009) - Nombre a los **pilares** del paradigma orientado a objetos.

(010) - ¿Qué es una **clase**?

(011) - ¿Qué es un **objeto**? ¿En qué se relacionan con las clases?

(012) - ¿Qué significa **instanciar** un objeto?

(013) - Explique qué es la **abstracción** en el contexto de programación orientada a objetos y cuál es su relación con las clases.

(014) - ¿Qué son los **atributos** o estado de un objeto?

(015) - ¿Qué son los **métodos** de un objeto?

[Módulo - 03] Miembros estáticos y de instancia:

(016) - ¿Qué es un **método estático**? ¿En qué se diferencia de los métodos de instancia (no estáticos)?

(017) - ¿Qué es una **clase estática**? ¿En qué se diferencia de las clases no estáticas?

(018) - ¿Puedo tener miembros estáticos en clases no-estáticas? ¿Puedo tener miembros no-estáticos en clases estáticas?

(019) - ¿Necesito instanciar un objeto de la clase para llamar a sus métodos estáticos?

(020) - ¿Cuántos valores distintos pueden existir para un campo estático y cuántos de uno no-estático cuando se crearon 5 instancias de la clase?

(021) - ¿Se puede usar el **operador "this"** dentro de un método estático? ¿Se puede acceder a miembros no-estáticos desde un método estático de la misma clase? ¿Se puede acceder a miembros estáticos desde un método de instancia?

(022) - ¿Se pueden declarar variables estáticas dentro de un método (locales)?

(023) - De dos ejemplos de métodos estáticos que pertenezcan a las clases de .NET Framework.

(024) - De un ejemplo de un método de instancia que pertenezca a las clases de .NET Framework.

[Módulo - 04] Namespaces y directivas:

(025) - ¿Qué es un **namespace** y cuál es su función principal?

(026) - ¿Puedo tener distintos namespaces dentro de un mismo proyecto o ensamblado de .NET?

(027) - ¿Para qué se usa la **directiva using**?

(028) - ¿Para qué se usa la **directiva alias**?

(029) - ¿Puedo declarar dos clases independientes/distintas con el mismo identificador dentro del mismo namespace? ¿Y en namespaces distintos?

[Módulo - 05] Objetos:

(030) - ¿Los objetos se crean en tiempo de diseño, de compilación o de ejecución?

(031) - ¿Cuántos objetos de tipo Alumno puedo tener en mi sistema? ¿Cuántas clases Alumno tengo en mi sistema? (Dentro de un mismo namespace).

(032) - Explique el **ciclo de vida** de un objeto. Detalle las funciones del **operador new**, del constructor y del **Garbage Collector**.

(033) - ¿En qué **segmento de memoria** se almacenan los tipos valor (value type) y en cuál los tipos por referencia (reference type)? ¿En cuál interfiere al Garbage Collector?

(034) - Compare y describa: **Destrucción determinista y no determinista**. Asocie con el concepto de variables y objetos.

(035) - ¿Cuál es la diferencia entre declarar, inicializar e instanciar un objeto?

[Módulo - 06] Constantes:

(036) - ¿Qué es una **constante**? ¿Se puede declarar constantes estáticas (static const)?

(037) - ¿Cuándo se asigna el valor a las constantes (tiempo de compilación o ejecución)? ¿Dos objetos del mismo tipo pueden tener distintos valores en una misma constante? Relacione con atributos de instancia y estáticos.

[Módulo - 07] Constructores:

(038) - ¿Qué es un **constructor**? ¿Cuál es su función?

(039) - ¿Qué es el **constructor por defecto**? ¿Qué sucede con el mismo cuando declaramos un constructor nuevo en la clase?

(040) - ¿Con qué valores se cargan los atributos cuando se llama al constructor por defecto?

(041) - ¿Qué es y para qué sirve un **constructor estático**? ¿En qué se diferencia su sintaxis de los constructores de instancia?

(042) - ¿Cuántas veces se puede llamar a un constructor estático? ¿Quién lo puede llamar?

(043) - ¿Se ejecutará primero un constructor estático o uno de instancia?

[Módulo - 08] Sobrecargas (Overload):

(044) - ¿Qué significa **sobrecargar** un método o constructor?

(045) - ¿Qué debe cambiar para que la sobrecarga de un método o constructor sea válida?

(046) - ¿La sobrecarga se resuelve en tiempo de ejecución o en tiempo de compilación? ¿Cómo se distingue a qué sobrecarga llamar?

(047) - ¿Se tiene en cuenta el nombre o identificador de los parámetros de entrada para una sobrecarga?

(048) - ¿Se tiene en cuenta el modificador de visibilidad para una sobrecarga?

[Módulo - 09] Sobrecarga de métodos:

(049) - ¿Los métodos pueden tener el mismo nombre que otros elementos de una misma clase? (atributos, propiedades, etc).

(050) - Mencione dos razones por las cuales se sobrecargan los métodos.

(051) - ¿Los métodos estáticos pueden ser sobrecargados?

(052) - ¿Agregar el modificador “static” sin cambiar los parámetros de entrada es una sobrecarga válida?

(053) - ¿Agregar un modificador “out” o “ref” en la firma del método sin cambiar nada más es una sobrecarga válida?

(054) - ¿Cambiar el tipo de retorno sin cambiar los parámetros de entrada es una sobrecarga válida?

(055) - Si tenemos distintas sobrecargas de un método, ¿cómo podemos reutilizar código?

[Módulo - 10] Sobrecarga de constructores:

(056) - ¿Para qué se utiliza el **operador “this()”**?

(057) - ¿Se pueden sobrecargar los constructores estáticos?

(058) - ¿Se puede llamar a un constructor estático con el operador “this()”?

(059) - ¿Se puede llamar a constructores de otras clases con el operador “this()”?

(060) - ¿Se puede sobrecargar un constructor privado?

[Módulo - 11] Sobrecarga de operadores:

(061) - ¿Qué es un **operador**? ¿En qué se diferencian un **operador unario** y un **operador binario**? De un ejemplo de cada uno.

(062) - ¿Qué varía en la sintaxis de la sobrecarga de operadores unarios y binarios?

(063) - ¿Se pueden sobrecargar los operadores de operación y asignación (+=, -=, *=, /=)? ¿Por qué?

(064) - ¿Cuál es la diferencia entre un **operador de conversión implícito** y uno **explícito**? (En finalidad, declaración y aplicación)

(065) - Los operadores de casteo “(T)x” no se pueden sobrecargar. ¿Cuál es la alternativa?

(066) - ¿Cuál es la diferencia entre **castear (casting)**, **convertir (converting)** y **parsear (parsing)**?

[Módulo - 12] Formularios:

(067) - ¿Los **formularios** son objetos?

(068) - ¿De qué clase heredan todos los formularios?

(069) - ¿Qué es una **partial class** o clase parcial?

(070) - ¿Puedo agregar parámetros de entrada a la clase del formulario? ¿Y sobrecargar el constructor? ¿Y declarar nuevos campos/propiedades?

(071) - ¿Cuál es la diferencia entre **Show()** y **ShowDialog()**?

(072) - ¿Qué es un **formulario MDI**? ¿Con qué propiedad indico que un formulario es un contenedor MDI? ¿Con qué propiedad del formulario hijo indico cuál es el formulario MDI padre?

(073) - Explique el **ciclo de vida** de los formularios asociándolo a sus eventos correspondientes.

[Módulo - 13] Arrays:

(074) - ¿Qué es un **array "jagged"**? ¿En qué valor se inicializan sus elementos?

(075) - ¿Los arrays son objetos?

(076) - ¿Qué significa que en C# los arrays son de **"base-cero"**?

(077) - Los arrays implementan la **interfaz IEnumerable**, ¿qué es lo que esto les permite hacer?

(078) - ¿Cómo se declara e instancia un **array multidimensional**?

(079) - Considerando que la **propiedad Length** devuelve la cantidad TOTAL de elementos de TODAS las dimensiones del array, ¿qué valor vamos a mostrar por consola en el siguiente código?:

a.
```
int[ , , ] a = new int[ 3, 2, 3 ];
int[ , ] b = new int[ 5, 4 ];
Console.WriteLine("Array Length = {0}", a.Length + b.Length);
```

[Módulo - 14] Strings:

(080) - ¿Es lo mismo declarar una variable como string (en minúscula) o como String (con la primera letra en mayúscula)? ¿Por qué?

(081) - ¿Se puede recorrer un string con un foreach? ¿Por qué?

(082) - ¿Qué significa que los strings son **inmutables**? ¿qué sucede en realidad cuando usamos métodos u operadores para modificar un string?

[Módulo - 15] Colecciones:

(083) - ¿Cuál es la diferencia entre las **colecciones** y las **matrices**?

(084) - ¿Cuál es la diferencia entre las **colecciones genéricas** y las **no genéricas**?

(085) - ¿Es necesario determinar el tipo de dato o realizar una conversión al recuperar un objeto de una colección genérica?

(086) - Describa los siguientes tipos de colecciones genéricas: **Dictionary**, **List**, **SortedList**.

(087) - ¿Qué son y cuál es la diferencia entre una **cola (queue)** y una **pila (stack)**? Asocie con los conceptos "**FIFO**" y "**LIFO**".

(088) - Describa los siguientes tipos de colecciones no genéricas: **ArrayList**, **Hashtable**.

(089) - ¿Se pueden ordenar directamente las colas y las pilas? ¿Por qué (piense en la función de dichas colecciones)? ¿Cuál es la alternativa?

(090) - ¿Cuál es la diferencia entre las colas y pilas genéricas y las colas y pilas no genéricas?

(091) - ¿Qué muestra el siguiente código?

```
short cantidad = 4;
Queue<int> cola = new Queue<int>();
Stack<int> pila = new Stack<int>();

for (int i = 0; i <= cantidad * 2; i += 2)
{
    cola.Enqueue(i);
}

foreach (int i in cola)
{
    pila.Push(i);
}

foreach (int i in pila)
{
    Console.Write("{0}, ", i);
}
```

a. 1, 2, 3, 4, 5,
b. 5, 4, 3, 2, 1,
c. 8, 6, 4, 2, 0,
d. 0, 2, 4, 6, 8,
e. 1, 3, 5, 7, 9,
f. 9, 7, 5, 3, 1,
g. Error en tiempo de ejecución. Marcar error.
h. Error en tiempo de diseño. Marcar error.

[Módulo - 16] Propiedades:

(092) - ¿Qué es y para qué sirve una **propiedad**?

(093) - ¿Para qué sirve el **descriptor de acceso "get"**?

(094) - ¿Para qué sirve el **descriptor de acceso "set"**? ¿Cuál es el papel de la **palabra clave "value"**?

(095) - ¿Cómo declaro una propiedad de **sólo lectura**?

(096) - ¿Cómo declaro una propiedad de **sólo escritura**?

[Módulo - 17] Enumerados:

(097) - ¿Qué es un **enumerado**? ¿Cuál es su función?

(098) - ¿Un enumerado sólo puede estar anidado dentro de una clase?

(099) - ¿Cuál es el primer valor numérico de un enumerado por defecto? ¿Se pueden sobrescribir los valores por defecto?

(100) - Indique los valores asociados a cada constante:

```
enum Day {Sat, Sun, Mon=15, Tue, Wed, Thu=2, Fri};
```

[Módulo - 18] Indexadores:

(101) - ¿Qué significa **indexar**?

(102) - ¿Qué permite un **indexador** (función)?

(103) - ¿Cuál es la diferencia a la hora de declarar un indexador y una propiedad?

(104) - ¿Los indexadores solo se pueden indexar por valores numéricos?

(105) - ¿Un indexador puede recibir más de un parámetro (ser multidimensionales)?

(106) - ¿Los indexadores pueden ser sobrecargados?

(107) - ¿Cuál es el papel de la palabra clave "this" en un indexador?

(108) - ¿Cuál es el papel de la palabra clave “value” en un indexador?

(109) - ¿Se pueden declarar indexadores estáticos? ¿Por qué?

[Módulo - 19] Encapsulamiento:

(110) - Defina **encapsulamiento / encapsulación** en el contexto del paradigma orientado a objetos.

(111) - Defina cada **nivel de ocultamiento / accesibilidad** de la programación orientada a objetos. Incluya la función del modificador “internal” en C#.

[Módulo - 20] Herencia:

(112) - ¿Qué es la **herencia** en el contexto de la programación orientada a objetos? ¿Cuál es su propósito?

(113) - ¿Qué nombre recibe la clase que hereda y qué nombre recibe la clase que es heredada?

(114) - ¿Qué significa que la herencia es **transitiva**?

(115) - ¿Se heredan los constructores?

(116) - ¿Se heredan los miembros private de la clase base?

(117) - ¿Qué es **herencia múltiple**? ¿Es posible en C#? ¿En qué se diferencia de la **herencia simple**?

(118) - ¿Una clase pública puede heredar de una clase privada?

(119) - ¿Qué es una **clase sellada (sealed)**?

(120) - ¿Una clase sellada puede heredar de otras clases? (Ser clase derivada)

(121) - ¿Cómo actúa el modificador “protected” en los miembros de la clase base para una clase derivada y cómo para una clase no-derivada? Relacionar la respuesta con los modificadores “public” y “private”.

(122) - ¿Qué pasa si la clase derivada no hace una llamada explícita a un constructor de la clase base? En esta situación, ¿qué pasa si la clase base declaró explícitamente un constructor con parámetros de entrada?

[Módulo - 21] Polimorfismo:

(123) - ¿Qué es el **polimorfismo** en el contexto de la programación orientada a objetos?

(124) - ¿Qué implica el polimorfismo basado en herencia?

[Módulo - 22] Sobreescritura de métodos (Override):

(125) - ¿Dónde reside la definición del método a sobrescribir? ¿Qué palabra clave se usa para definirlo?

(126) - ¿Dónde reside la implementación del método a sobrescribir? ¿Qué palabra clave se usa para implementarlo?

(127) - ¿Cuándo se resuelve la invocación? (Tiempo de ejecución o compilación)

(128) - ¿Cuáles son las diferencias entre **sobrecargar (overload) y sobrescribir (override)** un método? (Llenar la tabla)

Criterio	Sobrecargar / Overload	Sobrescribir / Override
Firma (Diferencias o no diferencias en las firmas)		
Ubicación (Misma clase / Clases diferentes)		
Tiempo de resolución (Compilación / Ejecución)		
Tipo objeto / Tipo Referencia (¿Qué determina cuál implementación se utilizará?)		

[Módulo - 23] Clases y miembros abstractos:

(129) - Si quiero declarar un método que pueda ser sobrescrito en las clases derivadas, ¿qué modificador debo usar?

(130) - Si quiero declarar un método que deba ser sobrescrito en las clases derivadas, ¿qué modificador debo usar?

(131) - ¿Qué es una **clase abstracta**? ¿Cuál es su función?

(132) - Las clases no-abstractas que derivan de una clase abstracta, ¿deben implementar todos sus métodos abstractos?

(133) - Las clases abstractas que derivan de una clase abstracta, ¿deben implementar todos sus métodos abstractos?

(134) - ¿Se pueden declarar **miembros abstractos** en clases no-abstractas?

(135) - ¿Para sobrescribir un método se debe heredar de una clase abstracta?

(136) - Llenar los campos de la siguiente tabla con SÍ o NO según corresponda.

Tipo de Clase	Puede heredar de otras clases (ser derivada)	Puede heredarse de ella (ser base)	Puede ser instanciada
normal (sin modificadores)			
abstract			
sealed			
static			