Cuestionario - Polimorfismo y clases abstractas

1. ¿Dónde reside la definición del método a sobrescribir? ¿Qué palabra clave se usa para definirlo?

   Reside en la clase padre, se utiliza virtual para definirlo

2. ¿Dónde reside la implementación del método sobrescrito? ¿Qué palabra clave se usa para implementarlo?

   Reside en la clase deriva/hija, se utiliza override

3. ¿La invocación de los métodos sobrescritos (override) se resuelve en tiempo de compilación o ejecución? ¿Y la de los métodos sobrecargados (overload)?

   Tiempo de ejecución, creo que en tiempo de compilacion

4. ¿Cambia la firma de los métodos cuando los sobrescribimos (override)? ¿Y cuando los sobrecargamos (overload)?

   No. Si debido a que debe recibir distintos parámetros para ser una sobrecarga

5. ¿Los métodos sobrescritos (override) se encuentran en la misma clase? ¿Y los métodos sobrecargados (overload)? ¿Qué es el polimorfismo en el contexto de la programación orientada a objetos?

   Creo que sí, Creo que sí. El polimorfismo es la habilidad de vincular objetos de diferentes tipos a un solo identificador en tiempo de ejecución.

6. ¿Todos los objetos en C# son polimórficos? ¿Por qué?

   En teoría si, debido a que cada objeto se lo puede vincular a otro objeto dependiendo de las características del mismo

7. Según el siguiente código, complete la tabla indicando:
   1. El tipo de la referencia (Persona o Profesor).
   2. El tipo del objeto en tiempo de ejecución (Persona o Profesor).
   3. La salida por consola.

```
public class Persona

{

    public virtual string Teach()

    {
```

```
        return "Una persona puede enseñar.";
    }
}

public class Profesor : Persona
{
    public override string Teach()
    {
        return "Un profesor puede enseñar en un colegio.";
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // Caso 1
        Persona persona = new Persona();
        Console.WriteLine(persona.Teach());

        // Caso 2
        Persona otraPersona = new Profesor();
        Console.WriteLine(persona.Teach());

        // Caso 3
```

```
        Profesor profesor = new Profesor();

        Console.WriteLine(persona.Teach());

    }

}
```

Objeto	Tipo de referencia	Tipo de la instancia en tiempo de ejecución	Texto de la salida por consola
persona	Persona	Persona	`"Una persona puede enseñar."`
otraPersona	Persona	Profesor	`"Una persona puede enseñar."`
profesor	Profesor	Profesor	`"Una persona puede enseñar."`

Clases y miembros abstractos

1. ¿Qué modificador debo utilizar si quiero declarar un método que **pueda** ser sobrescrito en las clases derivadas?

   Virtual

2. ¿Qué modificador debo utilizar si quiero declarar un método que **deba** ser sobrescrito en las clases derivadas?

   abstract

3. ¿Qué es una clase abstracta? ¿Cuál es su función?

   Una clase abstracta es un tipo de clase que nos da la posibilidad de hacer una definición particular que nos genere un modelo para una jerarquía de herencia

4. Las clases **no-abstractas** que derivan de una clase abstracta, ¿deben implementar todos sus métodos abstractos? ¿Por qué?

   Si, las clases derivadas deben realizar la implementación del método que se encuentra en la clase base(abstracta) debido a que la clase abstracta

5. Las clases **abstractas** que derivan de una clase abstracta, ¿deben implementar todos sus métodos abstractos? ¿Por qué?

No hace falta

6. ¿Se pueden declarar miembros abstractos en clases **no-abstractas**? ¿Por qué?

   No se puede

7. ¿Para sobrescribir un método se debe heredar de una clase abstracta? ¿Por qué?

   Si, se debe heredar de una clase abstracta para poder sobrescribir un metodo

8. Marque los campos de la siguiente tabla con SÍ o NO según si la afirmación es verdadera para el tipo de clase:

Tipo de clase	Puede heredar de otras clases (ser derivada)	Puede heredarse de ella (ser base)	Puede instanciarse
Clase normal (sin modificadores)	Si	Si	Si
Clase abstracta (abstract)	Si	Si	No
Clase sellada (sealed)	Si	No	Si
Clase estatica (static)	No	No	No

Herencia

- Qué es la herencia en el contexto de la programación orientada a objetos? ¿cuál es su propósito?

Es el mecanismo por el cual una clase permite heredar las características (atributos y métodos) de otra clase

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

Clase base, clase derivada

- Explique el principio de sustitución de Liskov.

Liskov fue una científica que propuso el concepto de que cada clase que hereda de otra, debe poder usarse como su padre sin necesidad de conocer la diferencia entre las dos

- ¿Qué significa que la herencia es transitiva?

Significa que si nosotros tenemos una clase C, la cual hereda de otra clase B, y al mismo tiempo, B hereda de otra clase A, C hereda de A

- ¿Se heredan los constructores?

No se pueden heredar constructores, pero si puedo llamar los costructores de la clase base, en las clase hija. Ejemplo

```
public Persona(string nombre, int edad)
{
        This.nombre = nombre;
        This.edad = edad;
}

Public Profesor(string nombre, int edad, double salario) :base(nombre,edad)
{
        This.salario = salario;
}
```

El :base() llamo al constructor de la clase padre

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

No, no se puede.

- ¿Qué es herencia múltiple? ¿es posible en C#? ¿en qué se diferencia de la herencia simple?

Se dice herencia múltiple cuando una clase derivada puede tener más de una clase base, adquiriendo los miembros de todos sus padres. Esto no es posible en c#. La diferencia principal es que la herencia simple, una clase derivada hereda de una sola clase base.

- ¿Una clase pública puede heredar de una clase privada?

No se puede, debido a su nivel de proteccion

- ¿Qué es una clase sellada (sealed)?

Una clase sellada (Sealed), implica que esa misma clase no podrá ser heredada (no pueda ser base)

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

Si, si puede

- ¿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”.

- ¿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?

Genera un error

- La clase `Alumno` hereda de `Persona`. ¿Una instancia de `Alumno` es también de tipo `Persona`? Justifique.

Si

- La clase `Alumno` hereda de `Persona`. ¿Se puede hacer `Persona persona = new Alumno()`? ¿por qué?

Si se puede hacer

- La clase `Alumno` hereda de `Persona`. ¿Se puede hacer `Alumno alumno = new Persona()`? ¿por qué?

No se puede

Cuestionario - Windows Forms

1. ¿Los formularios son objetos?
2. ¿En qué parte del código se instancia el primer formulario?
3. ¿De qué clase heredan/derivan todos los formularios?
4. ¿Qué es una partial class o clase parcial?
5. ¿Puedo agregar parámetros de entrada al constructor de la clase del formulario? ¿Puedo sobrecargarlo? ¿Por qué?
6. ¿Se pueden declarar nuevos campos/atributos/propiedades dentro del formulario? ¿Por qué?
7. ¿Cuál es la diferencia entre los métodos Show y ShowDialog?
8. ¿Qué es un formulario MDI? ¿Y uno SDI?
9. ¿Con qué propiedad indico que un formulario es un contenedor MDI?
10. ¿Modificando qué propiedad del formulario hijo indico cuál es el formulario MDI padre?
11. ¿Qué es un evento en Windows Forms? Mencione algunos eventos de los controles o de los formularios y explique cómo trabaja con ellos. ¿Cuál es el manejador del evento?
12. Explique el ciclo de vida de los formularios asociándolo a sus eventos.

Respuestas:

1. Los formularios son objetos que derivan de la clase Form perteneciente al namespace System.Windows.Forms.

2. Dentro del constructor por defecto, en InitializeComponent()

3. Derivan de la clase Form perteneciente al namespace System.Windows.Forms.

4. Es una clase única partida en dos partes, la visual (interfaz que ve el usuario) y la logica

5. Si, se puede tanto sobrecargar cómo agregar parámetros de entrada siempre y cuando siga teniendo la inicialización (InitializeComponent()), ya que es una clase de objeto como cualquier otra

6. Si, ya que un formulario es una clase de tipo objeto

7. El método Show () muestra un formulario de Windows en un estado no modal. El método ShowDialog () muestra una ventana en un estado modal y detiene la ejecución del contexto de llamada hasta que el método devuelva un resultado del formulario de ventanas abierto.

8. Un formulario MDI es el que se utiliza en un proyecto para acceder a múltiples formularios, generalmente es el formulario principal del proyecto.

Un formulario SDI es un formulario hijo, es decir, uno de los formularios que componen el proyecto pero que no es el principal.

9. Se indica con IsMdiContainer = true; dentro del load

10. Modificando la propiedad MdiParent, ejemplo: frmHijo.MdiParent = this

11. Un evento es una acción a la que puede responder, o que puede "controlar", en el código. Los eventos se pueden generar mediante una acción del usuario, como hacer clic con el mouse o presionar una tecla, mediante código de programa o por el sistema.

Eventos del punto 12

Al manejador de evento también se le denomina event handler. El manejador no es más que un método que se ejecuta al disparar el evento.

12. Initialize > Se hace referencia en el código al objeto Form.
    Load > Se hace referencia a las propiedades o controles de Form.

    Paint > Se "pinta" el formulario y sus controles

    Activated > El formulario recibe foco

    FormClosing > Permite cancelar el cierre.

    FormClosed > El formulario es invisible.

    Disposed > El objeto está siendo destruido. Se libera la memoria

¿Qué significa sobrecargar un método o constructor?

La sobrecarga de un método o constructor nos sirve para aplicar polimorfismo. Esto quiere decir que el método/constructor va a tener un comportamiento distinto dependiendo de los parámetros que le pasemos y la clase en la que se encuentren.

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

Los parámetros del método o constructor, en este último caso podemos tener 2 constructores sin parámetros, pero uno debe ser de clase (static) y otro de instancia.

¿La sobrecarga se resuelve en tiempo de ejecución o en tiempo de compilación?

Ocurre sobre tiempo de compilación. El compilador selecciona el método apropiado mediante un análisis de su firma.

¿Cómo se distingue a qué sobrecarga llamar?

Se distingue analizando la firma del método.

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

No.

¿Se tiene en cuenta el modificador de visibilidad para una sobrecarga?

Depende, en los constructores no pero en un método heredado sobrecargado sí. En el caso de sobrecargar un operador debe ser público siempre.

¿Se tiene en cuenta el tipo de retorno para una sobrecarga?

No. (Si trata de cuando se hereda de un método abstracto deben coincidir.)

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

No, nos saldrá que la clase ya tiene una definición con ese nombre.

Mencione dos razones por las cuales se sobrecargan los métodos.

Mejora la usabilidad, la productividad y la legibilidad de nuestro código.

Suple la necesidad de poder reutilizar el mismo nombre para un método que hace cosas parecidas con distintos parámetros.

¿Los métodos estáticos pueden ser sobrecargados?

Si pueden ser sobrecargados.

¿Agregar el modificador "static" sin cambiar los parámetros de entrada es una sobrecarga válida?

No.

¿Agregar un modificador "out" o "ref" en la firma del método sin cambiar nada más es una sobrecarga válida?

Si, lo son.

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

No.

¿Se pueden sobrecargar los constructores estáticos? ¿Por qué?

No, no pueden sobrecargarse ya que se llaman una sola vez en runtime.

¿Se puede llamar a un constructor estático con el operador "this()"?

No, solo puede ser llamado por el CLR

¿Se puede llamar a constructores de otras clases con el operador "this()"?

No, en el caso de ser la clase padre se debe usar base().

¿Se puede sobrecargar un constructor privado?

Si, se puede.

¿Qué es un operador?

Un operador es un elemento de programa que se aplica a uno o varios operandos en una expresión o instrucción.

¿En qué se diferencian un operador unario y un operador binario? De un ejemplo de cada uno.

Los operadores unarios realizan una acción con un solo operando

¡var (negación del valor "var")

Los operadores binarios realizan acciones con dos operandos. En una expresión compleja (dos o más operandos), el orden de evaluación depende de las reglas de precedencia.

Ej. 8/2 (Operación aritmética de división donde 8 es el dividendo y 2 el divisor)

¿Qué varía en la sintaxis de la sobrecarga de operadores unarios y binarios?

Ambos deben ser públicos y estáticos, cambia la cantidad de parámetros necesarios. Siendo 1 parámetro para el caso de operadores unarios y 2 para los binarios. En ambos se debe usar la palabra reservada operator.

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

Si, estos son sobrecargados automáticamente cuando su operador binario correspondiente es sobrecargado.

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

El casting implícito se realiza de forma automática al asignar un valor de un tipo a otro compatible.

Las conversiones implícitas no requieren que se invoque una sintaxis especial

Declaracion:

[acceso] static implicit operator nombreTipo(Tipo a)

{

  //…

}

En el casting explicito es tarea del programador especificar el nuevo tipo al que se va a transformar el dato. Se escribe de forma explicita entre paréntesis delante del dato.

Las conversiones explícitas requieren una expresión de conversión (tipo).

Puede haber pérdida de información

Declaracion:

[acceso] static explicit operator nombreTipo(Tipo a)

{

  //…

}

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

No encontré esta

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

Casting	**Converting**	**Parsing**
En la conversión de tipos, un programador convierte un	**Mientras que en la conversión de tipos, un**	**Cuando parseamos estamos convirtiendo, sin embargo**

tipo de datos en otro tipo de datos mediante un operador de conversión.	**compilador convierte un tipo de datos en otro tipo de datos.**	**nos va a arrojar excepciones como null exception cuando se trata de un caso erróneo**