Guía de Estudio : Clase 04

1. ¿Qué significa **sobrecargar** un método o constructor?

Sobrecargar se lo denomina cuando tenemos dos o mas métodos en una clase que se llaman igual.

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

Se deben cambiar 3 cosas, el número, el tipo y el orden de los parámetros (cambiar la firma del método).

3. ¿La sobrecarga se resuelve en tiempo de ejecución o en tiempo de compilación? ¿Cómo se distingue a qué sobrecarga llamar?

Se resuelve en tiempo de compilación, donde el compilador sabe a que método se está llamando consecuentemente de analizar los parámetros que se están pasando.

4. ¿Se tiene en cuenta el nombre o identificador de los parámetros de entrada para una sobrecarga?  
No, tanto los nombres como los tipos de retorno del método no son tenidos en cuenta.

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

El nivel de visibilidad debe ser igual o menos restrictivo al que sobrescribe.

**Sobrecarga de Métodos:**

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

No, en el caso de que se lo desee así, se debe cambiar alguna letra por mayúscula o minúscula.

7. Mencione dos razones por las cuales se sobrecargan los métodos.

Si se necesita métodos similares que requieran parámetros diferentes, y si se quiere añadir funcionalidad al código existente, poniendo un único nombre y pasando los parámetros necesarios.

8. ¿Los métodos estáticos pueden ser sobrecargados?

Si.

9. ¿Agregar el modificador “static” sin cambiar los parámetros de entrada es una sobrecarga válida?

No, ya que un método estático pertenece a la clase, se invocaría para instanciar la misma y no un objeto.

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

Si. Siempre y cuando no sean de igual tipo y cantidad, por ej. un método con (out int numero) y otro con (int numero) es válido.

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

Correcto.

12. Si tenemos distintas sobrecargas de un método, ¿cómo podemos reutilizar código?

//Duda.

**Sobrecarga de Constructores:**

13. ¿Se pueden sobrecargar los constructores estáticos?

No, ya que no recibe parámetros ni tiene modificadores de acceso, es llamado automáticamente en tiempo de ejecución la primera vez que se inicializa un campo estático.

14. ¿Se puede llamar a un constructor estático con el **operador “this()”** ?

No.

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

No, solo a los de la misma clase que tenga diferente firma.

16. ¿Se puede sobrecargar un constructor privado?

Si, pero debe ser accesible desde otro constructor usando el operador this().

**Sobrecarga de Operadores:**

17. ¿Qué es un **operador?** ¿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. Los operadores binarios realizan acciones con dos operandos.

Ejemplo de operador unario: ++, --, !, &, etc.

Ejemplo de operador binario: +, -, \*, /, etc.

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

La sobrecarga de operadores unarios puede no recibir un argumento, salvo que esta sea GLOBAL, lo cual debe recibir uno de tipo de clase.

En cambio, los binarios deben recibir un argumento de tipo de clase, si es GLOBAL debe recibir dos, y uno de ellos debe ser de tipo de clase como mínimo.

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

Un tipo definido por el usuario puede sobrecargar los operadores unarios (++, --, + y -), los operadores binarios (\*, /, %, + y -) y los operadores aritméticos. Cuando se sobrecarga un operador binario, también se sobrecarga de forma implícita el operador de asignación compuesta correspondiente. Un tipo definido por el usuario no puede sobrecargar de forma explícita un operador de asignación compuesta.

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

Los operadores de conversión permiten hacer compatibles tipos que antes no lo eran.

Se declaran igual, cambiando la palabra explicit por implicit (o viceversa). Cabe destacar que deben ser STATIC.

[mod acceso] static implicit operator nombreTipo(Tipo a)

{

//…

}

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

Se deben definir nuevos operadores de conversión explicita/implícita.

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

**Casting** es cuando tomas una variable de un tipo y la cambias a otro tipo SIN alterar el valor de la variable.

Se puede hacer solo si los tipos son compatibles.

String str=”Hola”;

Object obj = str;

String str2 = (string) o;

**Converting** es similar a casting, con la diferencia de que puede ser es entre dos tipos no compatibles. Puede haber pérdida de información como el ejemplo.

Double d=5.5;

Int i = (int) d;

**Parsing** es cuando interpretar y convertir un string a un tipo diferente entendiendo su contenido, por ejemplo conversir el string “123” al numero 123., o el string “Sabado 22 de septiembre” a tipo DateTime.