

Programación II

C

Clase 3

Maximiliano Neiner

Modificada por: Federico Dávila

Temas a Tratar

- ❖ Objetos
- ❖ Constructores

Temas a Tratar

Objetos

- Características
- Sintaxis
- Ciclo de vida
- Objetos vs. Variables
- Constantes

Constructores

¿Qué es un objeto?

- ✖ Los objetos son instancias de una clase.
- ✖ Son creados en tiempo de ejecución.
- ✖ Poseen Comportamiento (métodos) y Estado (atributos).
 - Comportamiento:
 - Le permite realizar tareas específicas.
 - Estado:
 - Le permite almacenar información fija o variable.
- ✖ Para acceder tanto a los atributos como a los métodos se utiliza el operador punto (.).
- ✖ Para crear un objeto se necesita la palabra reservada NEW.

¿Qué es un objeto?



Temas a Tratar

Objetos

- Características
- Sintaxis
- Ciclo de vida
- Objetos vs. Variables
- Constantes

Constructores

Objetos - Sintaxis Declaración

```
Nombre_Clase nombre_objeto;
```

Dónde:

- ❖ **Nombre_Clase:** Es el identificador de la clase o del tipo de objeto al que se referirá.
- ❖ **nombre_objeto:** Es el nombre asignado a la instancia de tipo Nombre_Clase.
- ❖ **Nota:** Los nombres de los objetos deben estar en minúsculas. Ejemplo: casa, miCasa.

Objetos - Sintaxis Inicialización

```
nombre_objeto = new Nombre_Clase();
```

Dónde:

- ❖ **Nombre_Clase():** Es el *constructor* del objeto y no el *tipo* de objeto.
- ❖ Una vez inicializado el objeto se puede utilizar para manipular sus atributos y llamar a sus métodos.
- ❖ **Nota:** La declaración e inicialización de un objeto se puede hacer en una misma instrucción. Ejemplo:

```
Nombre_Clase nombre_objeto = new Nombre_Clase();
```

Temas a Tratar

- ❖ **Objetos**
 - Características
 - Sintaxis
 - Ciclo de vida
 - Objetos vs. Variables
 - Constantes
- ❖ **Constructores**

Objetos – Ciclo de vida

❖ Creación del objeto

- Se usa **new** para asignar memoria.
- Se usa un constructor para inicializar un objeto en esa memoria.

❖ Utilización del objeto

- Llamadas a métodos y atributos.

❖ Destrucción del objeto

- Se pierde la referencia en memoria, ya sea por finalización del programa, cambio o eliminación de la variable, etc.
- El Garbage Collector liberará memoria cuando lo crea necesario.

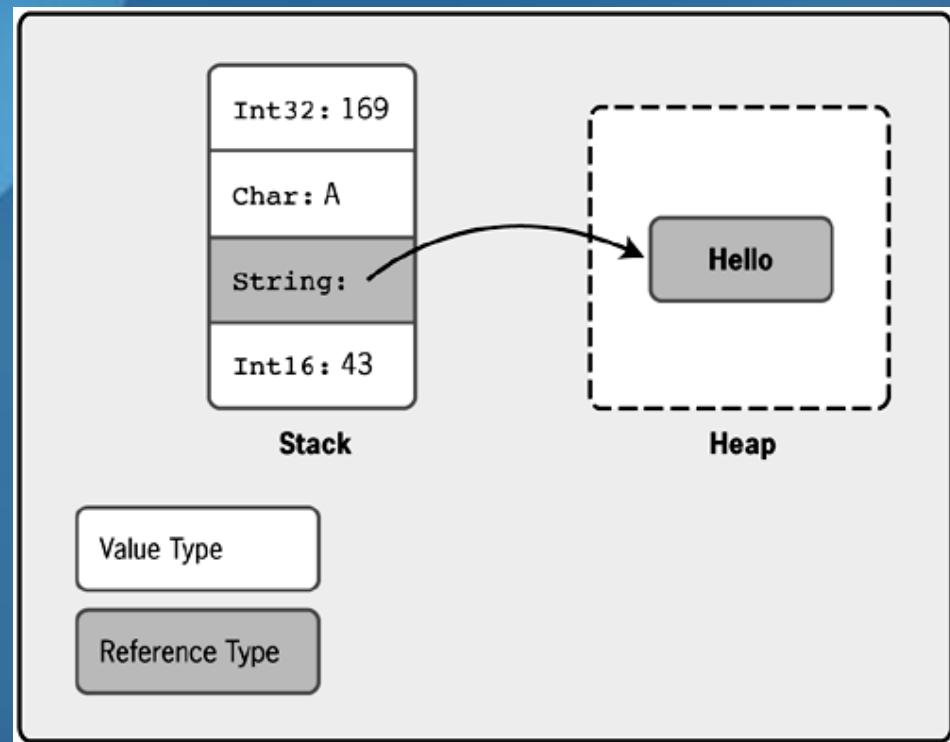
Garbage Collector

- ❖ En .NET el Garbage Collector será el encargado de liberar memoria.
- ❖ Cada vez que creamos un nuevo objeto, el CLR (Common Language Runtime) asigna memoria desde la porción gestionada (Heap)
- ❖ Eventualmente el Garbage Collector liberará memoria de objetos sin referencia.

<https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/>

La memoria y los Tipos de Datos

- El CLR administra dos segmentos de memoria: **Stack (Pila)** y **Heap (Montón)**.
- El **Stack** es liberado automáticamente y el **Heap** es administrado por el **GC (Garbage Collector)**.
- Los tipos **VALOR** se almacenan en el **Stack**.
- Los tipos **REFERENCIA** se almacenan en el **Heap**.



Temas a Tratar

Objetos

- Características
- Sintaxis
- Ciclo de vida
- Objetos vs. Variables
- Constantes

Constructores

Objetos vs. Variables

- ❖ El tiempo de vida de un valor local está vinculado al ámbito en el que está declarada.
 - Tiempo de vida corto (en general).
 - Creación y destrucción deterministas.

- ❖ El tiempo de vida de un objeto dinámico no está vinculado a su ámbito.
 - Tiempo de vida más largo.
 - Destrucción no determinista.

Variables

Creación y destrucción deterministas

- Una variable local se crea en el momento de declararla y se destruye al final del ámbito en el que está declarada. El punto inicial y el punto final de la vida del valor son deterministas; es decir, tienen lugar en momentos conocidos y fijos.

Tiempos de vida muy cortos por lo general

- Un valor se declara en alguna parte de un método y no puede existir más allá de una llamada al método. Cuando un método devuelve un valor, lo que se devuelve es una copia del valor.

Ámbito de una Variable

- Los valores locales son variables que se asignan en la pila (stack) y no en el Managed Heap. Esto significa que, si se declara una variable cuyo tipo es uno de los primitivos (como `int`, `enum` o `bool`), no es posible usarla fuera del ámbito en el que se declara.
- Ejemplo:

```
for(int i = 0; i < 10; i++)  
{  
    Console.WriteLine("i = {0}", i);  
}  
// i no es válido fuera del bloque for
```

Objetos

• Destrucción no determinista

- Un objeto aparece cuando se crea, pero, a diferencia de un valor, no se destruye al final del ámbito en el que se crea. La creación de un objeto es determinista, pero no así su destrucción. No es posible controlar exactamente cuándo se destruye y libera memoria para un objeto.

• Tiempos de vida más largos

- Puesto que el tiempo de vida de un objeto no está vinculado al método que lo crea, un objeto puede existir mucho más allá de una llamada al método.

Objetos

- ❖ El tiempo de vida de un objeto no está vinculado al ámbito en el que se crea. Los objetos se inicializan en memoria del Managed Heap mediante el operador **new**.
- ❖ Ej.: se inicializa con **new Example()**, cuyo ámbito no se acaba con el de ej.

```
class Example{  
    public void Metodo(int limite){  
        for(int i = 0; i < limite; i++) {  
            Example ej = new Example();  
        }  
    }  
}
```

Destrucción de objetos

- ✖ No es posible destruir objetos de forma explícita
 - Esto se debe a que una función de eliminación explícita es una importante fuente de errores en otros lenguajes.

- ✖ Los objetos se destruyen por recolección de basura (Garbage Collector).
 - Busca objetos inalcanzables y los destruye.
 - Los convierte de nuevo en memoria binaria no utilizada.
 - Normalmente lo hace cuando empieza a faltar memoria o cuando finaliza la aplicación.

Temas a Tratar

Objetos

- Características
- Sintaxis
- Ciclo de vida
- Objetos vs. Variables
- Constantes

Constructores

Constantes

- ❖ Una constante es otro tipo de campo.
- ❖ Contiene un valor que se asigna cuando se compila el programa y nunca cambia.
- ❖ Las constantes se declaran con la palabra clave **const**; son útiles para que el código sea más legible.

```
const int speedLimit = 55;
```

Temas a Tratar

- ❖ Objetos
- ❖ Constructores
 - Generalidades
 - Tipos

Generalidades (1/2)

- ✖ Los constructores son métodos especiales que se utilizan para inicializar objetos al momento de su creación.
- ✖ En C#, la única forma de crear un objeto es mediante el uso de la palabra reservada **new** para adquirir y asignar memoria.
- ✖ Aunque no se escriba ningún constructor, existe uno por defecto que se usa cuando se crea un objeto a partir de un tipo referencia.
- ✖ Los constructores llevan el mismo nombre de la clase.

Generalidades (2/2)

- ✖ Lo único que **new** hace es adquirir memoria binaria sin inicializar, mientras que el solo propósito de un constructor de instancia es inicializar la memoria y convertirla en un objeto que se pueda utilizar. En particular, **new** no participa de ninguna manera en la inicialización y los constructores de instancia no realizan ninguna función en la adquisición de memoria.
- ✖ Aunque **new** y los constructores de instancia realizan tareas independientes, un programador no puede emplearlos por separado. De esta forma, C# contribuye a garantizar que la memoria está siempre configurada para un valor válido antes de que se lea (a esto se le llama *asignación definida*).

Temas a Tratar

- ❖ Objetos
- ❖ Constructores
 - Generalidades
 - Tipos

Constructores - Tipos

- Hay dos tipos de constructores:
 - Constructores de instancia: que inicializan objetos (atributos NO estáticos).
 - Constructores estáticos: que son los que inicializan clases (atributos estáticos).

Constructores por Defecto (1/2)

Características de un constructor por defecto

- Acceso público.
- Mismo nombre que la clase.
- No tiene tipo de retorno (ni siquiera **void**).
- No recibe ningún argumento.
- Inicializa todos los campos a **cero, false o null**.
- Sintaxis del constructor:

```
class MiClase
{
    public MiClase()
    {
        //Inicializar atributos de instancia aquí
    }
}
```

Ejemplo # 1:

```
class Fecha
{
    private int _aaaa;
    private int _mm;
    private int _dd;
}

class Test
{
    public static void Main()
    {
        Fecha cuando = new Fecha();
        ...
    }
}
```

Constructores por Defecto (2/2)

- ❖ Si el constructor por defecto generado por el compilador no es adecuado, lo mejor es que escribamos nuestro propio constructor.
- ❖ Podemos escribir un constructor que contenga únicamente el código necesario para inicializar campos con valores distintos de cero. Todos los campos que no estén inicializados en este constructor conservarán su inicialización predeterminada a cero, false o null.

Ejemplo # 2:

```
class Fecha
{
    private int _aaaa;
    private int _mm;
    private int _dd;

    public Fecha()
    {
        // Fecha histórica
        this._aaaa = 1986;
        this._mm = 4;
        this._dd = 6;
    }
}
```

Constructores que reciben argumentos

- ❖ Como se mencionó anteriormente, los constructores son métodos, por lo tanto pueden recibir valores pasados como argumentos.
- ❖ Estos argumentos no escapan a la sintaxis de los parámetros de los métodos.
`tipo nombre_argumento`
- ❖ El número de argumentos en principio no tiene límites.

Ejemplo # 3:

```
class Fecha {  
    private int _aaaa;  
    private int _mm;  
    private int _dd;  
  
    public Fecha(int anio, int mes, int dia) {  
        this._aaaa = anio;  
        this._mm = mes;  
        this._dd = dia;  
    }  
}  
class Test {  
    static void Main() {  
        Fecha cuando = new Fecha(1986, 4, 6);  
        ...  
    }  
}
```

Constructores Estáticos

- ❖ Son los encargados de inicializar clases.
- ❖ Sólo inicializará los atributos estáticos.
- ❖ No debe llevar modificadores de acceso.
- ❖ Utilizan la palabra reservada **static**.
- ❖ No pueden recibir parámetros.

Ejemplo

```
class Clase {  
    private static int _var1;  
    private static int _var2;  
    private int _var3;  
  
    static Clase() {  
        _var1 = 3;  
        _var2 = 5;  
    }  
}  
class Test {  
    static void Main() {  
  
        new Clase();  
    }  
}
```