Microsoft
.NET
PAV 2023

La Programación Básica en Entorno Visual (I)

UNIDAD N° 2

(Parte 2/3)

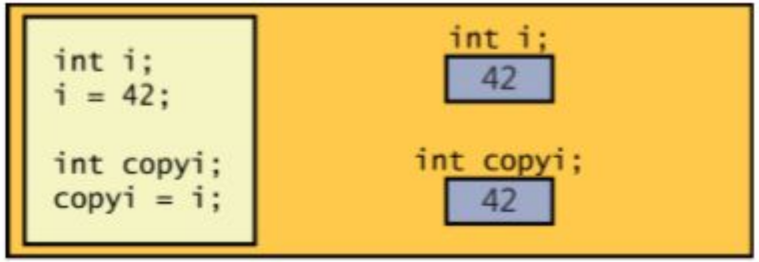
2023

Temas

- Trabajando con variables, operadores y expresiones. Escribir métodos y definir ámbitos.
- Usando operadores de asignación compuestos y declaraciones de iteración.
- Manejando errores y excepciones.
- Repasando: tipos por valor y referencia, arreglos.
- Creando y manejando clases y objetos. Trabajando con herencia.
- Implementando propiedades para el acceso a campos.
- Creando tipos con enumeraciones y estructuras.
- Usando arrays. Entendiendo los parámetros de arrays.
- Trabajando con interfaces y definiendo clases abstractas.
- Usando y enumerando colecciones. Consultando datos en memoria con lenguaje de consulta.

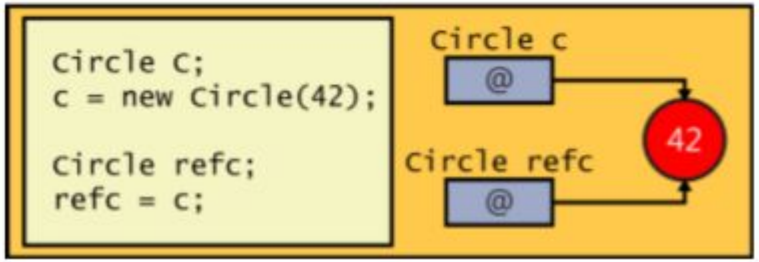
CHAPTER 8

Understanding values and references



```
int i;
i = 42;

int copyi;
copyi = i;
```



```
Circle C;
c = new Circle(42);

Circle refc;
refc = c;
```

Ejercicio N° 11

Understanding null values and nullable types

Inicialización

```
int i = 0;
double d = 0.0;
```

```
Circle c = new Circle(42);
```

Para definir una copia…

```
Circle c = new Circle(42);
Circle copy = new Circle(99);  // Some random value, for initializing copy
...
copy = c;                      // copy and c refer to the same object
```

¿Qué hacer si no deseo crear un nuevo objeto?

Understanding null values and nullable types

```
Circle c = new Circle(42);
Circle copy;                    // Uninitialized !!!
...
if (copy == // only assign to copy if it is uninitialized, but what goes here?)
{
    copy = c;                   // copy and c refer to the same object
    ...
}
```

```
Circle c = new Circle(42);
Circle copy = null;         // Initialized
...
if (copy == null)
{
    copy = c;               // copy and c refer to the same object
    ...
}
```

Para los tipos por referencia

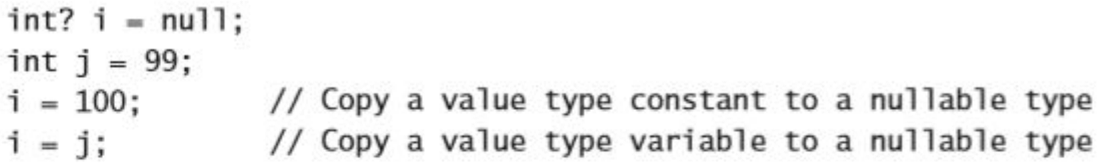
Y en los tipos por valor?

```
int i = null; // illegal
```

```
int? i = null; // legal
```

Understanding null values and nullable types

Ejemplo. Es posible el siguiente código?



```
int? i = null;
int j = 99;
i = 100;        // Copy a value type constant to a nullable type
i = j;          // Copy a value type variable to a nullable type
```

OK

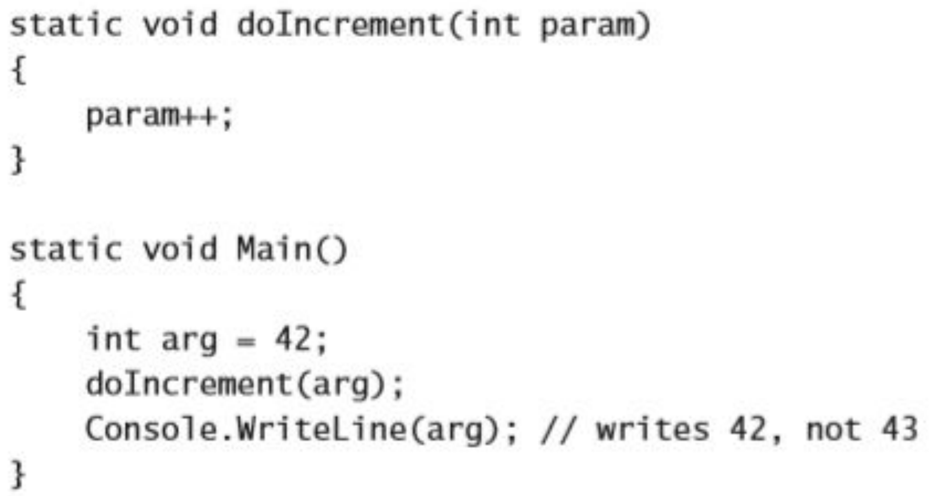
```
j = i;          // Illegal
```

NO

Understanding the properties of nullable types

```
int? i = null;
...
if (!i.HasValue)
{
   // If i is null, then assign it the value 99
    i = 99;
}
else
{
    // If i is not null, then display its value
    Console.WriteLine(i.Value);
}
```

Using *ref* and *out* parameters



```
static void doIncrement(int param)
{
    param++;
}

static void Main()
{
    int arg = 42;
    doIncrement(arg);
    Console.WriteLine(arg); // writes 42, not 43
}
```

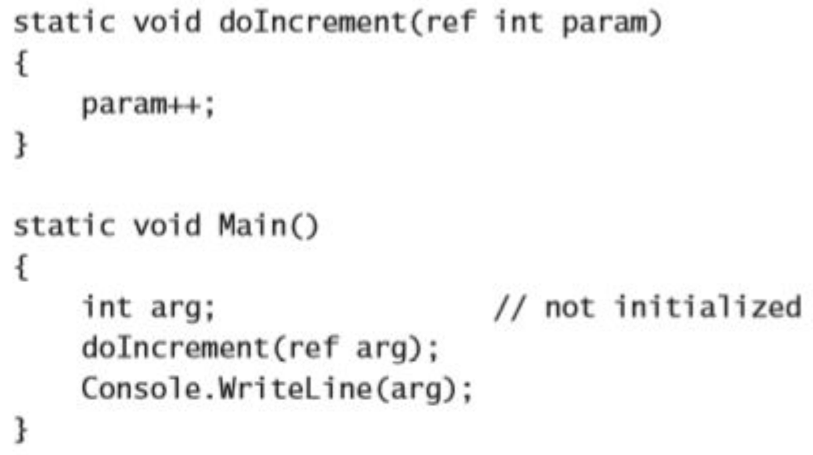
Se realiza una copia del argumento en el parámetro, aunque sea un tipo por referencia.

Using *ref* and *out* parameters

```
static void doIncrement(ref int param) // using ref
{
    param++;
}

static void Main()
{
    int arg = 42;
    doIncrement(ref arg);    // using ref
    Console.WriteLine(arg);  // writes 43
}
```

Restricción: el valor debe estar inicializado



```
static void doIncrement(ref int param)
{
    param++;
}

static void Main()
{
    int arg;                    // not initialized
    doIncrement(ref arg);
    Console.WriteLine(arg);
}
```

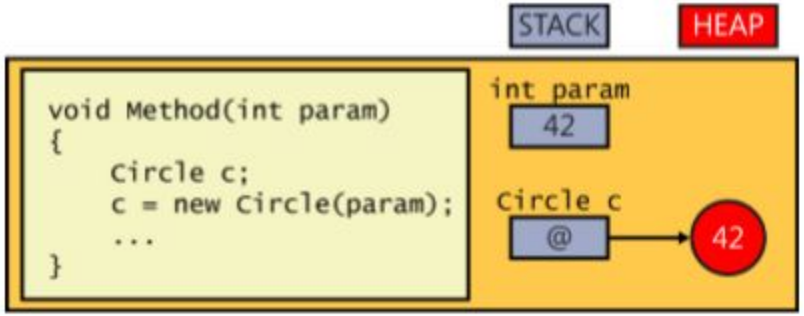
Using *ref* and *out* parameters

```
static void doInitialize(out int param)
{
    param = 42;
}

static void Main()
{
    int arg;                // not initialized
    doInitialize(out arg);  // legal
    Console.WriteLine(arg); // writes 42
}
```

Using the stack and the heap

```
void Method(int param)
{
    Circle c;
    c = new Circle(param);
    ...
}
```

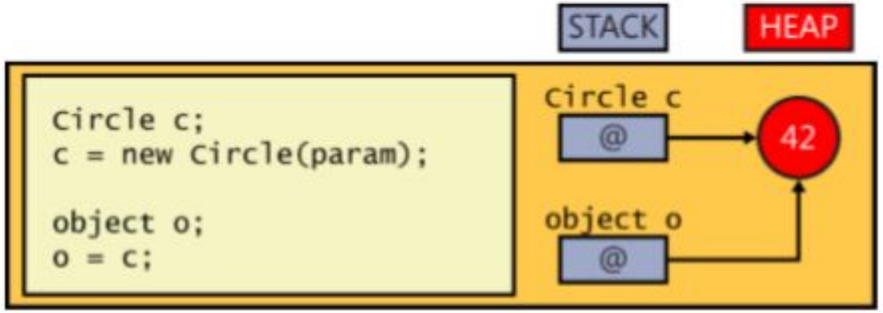


```
void Method(int param)
{
    Circle c;
    c = new Circle(param);
    ...
}
```

The *System.Object* class

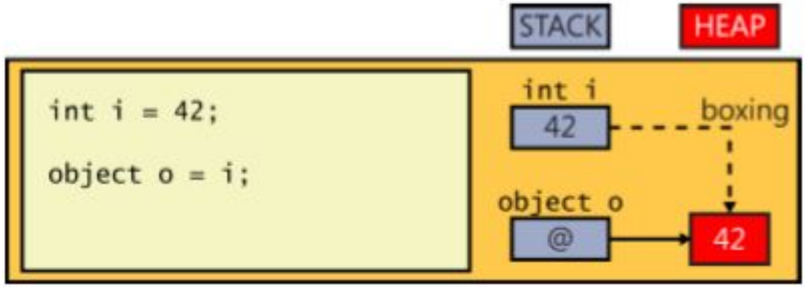
```
Circle c;
c = new Circle(42);
object o;
o = c;
```

The following diagram illustrates how the variables *c* and *o* refer to the same item on the heap.



Boxing

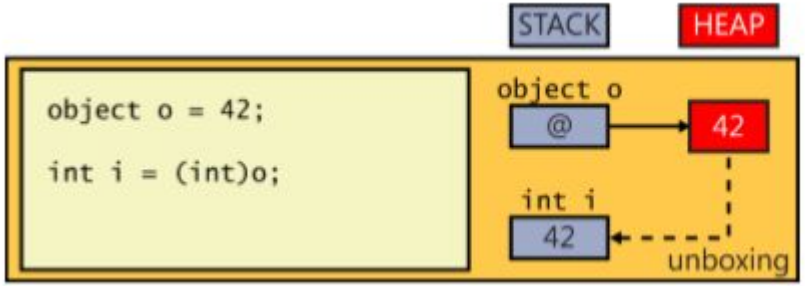
```
int i = 42;
object o = i;
```



```
int x = o;
```

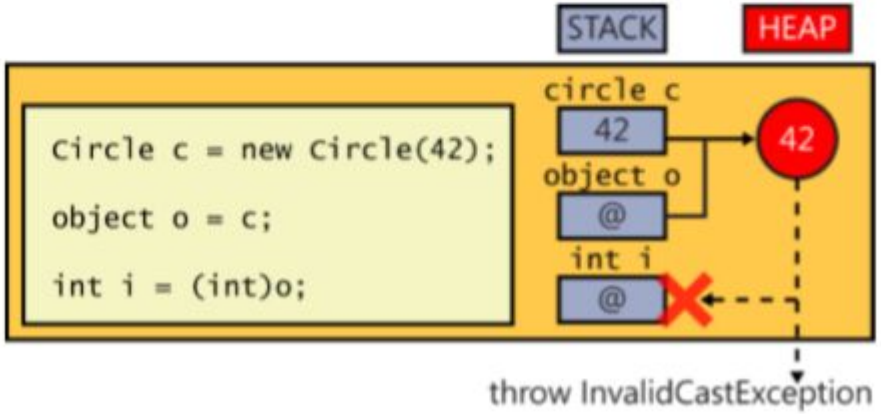
Unboxing

```
int i = 42;
object o = i; // boxes
i = (int)o;   // compiles okay
```


```
object o = 42;

int i = (int)o;
```

```
Circle c = new Circle(42);
object o = c;
int i = (int)o;
```



Casting data safely

The *is* operator

```
WrappedInt wi = new WrappedInt();
...
object o = wi;
if (o is WrappedInt)
{
    WrappedInt temp = (WrappedInt)o; // This is safe; o is a WrappedInt
    ...
}
```

The *as* operator

```
WrappedInt wi = new WrappedInt();
...
object o = wi;
WrappedInt temp = o as WrappedInt;
if (temp != null)
{
    ...  // Cast was successful
}
```

CHAPTER 7

Creating and managing classes and objects

Trabajando con Clases

- Constructores (sobrecarga)
- Clases parciales
- Métodos y datos estáticos
    - Métodos estáticos
    - Campos compartidos
    - Campos estáticos con **const**
    - Clases estáticas
- Clases anónimas

CHAPTER 12

Working with inheritance

Trabajando con Herencia

- Uso
- Llamar al constructor de la clase base
- Asignar clases
- Declarar nuevos métodos
- Declarar métodos con *virtual*
- Declarar métodos con *override*
- Acceso con *protected*
- Crear métodos de extensión

CHAPTER 15

Implementing properties to access fields

What are properties?

```
AccessModifier Type PropertyName
{
    get
    {
        // read accessor code
    }

    set
    {
        // write accessor code
    }
}
```

Generating automatic properties

```
class Circle
{
    public int Radius{ get; set; }
    ...
}
```

```
class Circle
{
    private int _radius;
    public int Radius{
        get
        {
            return this._radius;
        }
        set
        {
            this._radius = value;
        }
    }
    ...
}
```

Initializing objects by using properties

```
public class Triangle
{
    private int side1Length;
    private int side2Length;
    private int side3Length;

    // default constructor - default values for all sides
    public Triangle()
    {
        this.side1Length = this.side2Length = this.side3Length = 10;
    }
    // specify length for side1Length, default values for the others
    public Triangle(int length1)
    {
        this.side1Length = length1;
        this.side2Length = this.side3Length = 10;
    }

    // specify length for side1Length and side2Length,
    // default value for side3Length
    public Triangle(int length1, int length2)
    {
        this.side1Length = length1;
        this.side2Length = length2;
        this.side3Length = 10;
    }

    // specify length for all sides
```

Initializing objects by using properties

```
public class Triangle
{
    private int side1Length = 10;
    private int side2Length = 10;
    private int side3Length = 10;

    public int Side1Length
    {
        set { this.side1Length = value; }
    }

    public int Side2Length
    {
        set { this.side2Length = value; }
    }

    public int Side3Length
    {
        set { this.side3Length = value; }
    }
}
```

Initializing objects by using properties

```
Triangle tri1 = new Triangle { Side3Length = 15 };
Triangle tri2 = new Triangle { Side1Length = 15, Side3Length = 20 };
Triangle tri3 = new Triangle { Side2Length = 12, Side3Length = 17 };
Triangle tri4 = new Triangle { Side1Length = 9, Side2Length = 12,
                               Side3Length = 15 };
```

CHAPTER 9

Creating value types with enumerations and structures

Trabajando con Enumeraciones

- Declaración
- Uso
- Asignar valores específicos
- Definir con tipos especiales

Working with structures

```
struct Time
{
    public int hours, minutes, seconds;
}
```

Understanding structure and class differences

Question	Structure	Class
Is this a value type or a reference type?	A structure is a value type.	A class is a reference type.
Do instances live on the stack or the heap?	Structure instances are called values and live on the stack.	Class instances are called objects and live on the heap.
Can you declare a default constructor?	No.	Yes.
If you declare your own constructor, will the compiler still generate the default constructor?	Yes.	No.
If you don't initialize a field in your own constructor, will the compiler automatically initialize it for you?	No.	Yes.
Are you allowed to initialize instance fields at their point of declaration?	No.	Yes.