Alejandro García Guillén

Luis Alberto Alonso Hernández

Laboratorio de Programación Visual

Programación Orientada a Objetos en C#

Primer Parcial

3 de Enero del 2015

Programación Orientada a Objetos en C#

En C# los elementos que definen a una clase son:

- Atributos.
- Métodos.
- Constructores.

C# añade dos tipos nuevos de declaraciones:

- Propiedades:
  - Representan características de los objetos que son accedidas como si fueran atributos.
  - Ocultan el uso de métodos get/set.
  - Una propiedad puede representar un atributo calculado.
- Eventos:
  - Notificaciones que envía un objeto a otros objetos cuando se produce un cambio de estado significativo.

Los miembros de una clase pueden ser de instancia (por defecto) o de clase, utilizando el modificador static.

Los atributos de sólo lectura se marcan utilizando el modificador readonly.

- readonly Persona titular;

Las constantes se declaran const.

- private const int MAX = 20;

Esta permitida la inicialización de los atributos en la declaración.

- private double saldo = 100;

Los atributos no inicializados en la declaración o en los constructores toman el valor por defecto de su tipo de datos.

El nivel de visibilidad se especifica para cada declaración:

- public: visible para todas las clases.
- private: visible sólo para la clase.
- protected: visibilidad para la clase y los subtipos.
- internal: visibilidad para el “ensamblado”.
- protected internal: visibilidad para la clase y subtipos dentro del mismo ensamblado.

Por defecto, las declaraciones en una clase son privadas.

Un espacio de nombres (namespace) es un mecanismo para agrupar un conjunto de declaraciones de tipos relacionadas.

C# define un nivel de visibilidad entre los tipos que forman parte del mismo “ensamblado”, visibilidad internal.

Ensamblado: unidad de empaquetado de software en la plataforma .NET

- Un fichero ejecutable es un ensamblado.
- Un ensamblado es un componente software.

Se distinguen dos tipos de datos:

- Tipos con semántica referencia: clases, interfaces, arrays y “delegados”. Aceptan valor null.
- Tipos con semántica por valor: tipos primitivos, enumerados y estructuras.

Declaración de un enumerado:

- enum cuenta { OPERATIVA, INMOVILIZADA, NUMEROS_ROJOS}

Si no se inicializa un enumerado, toma como valor por defecto la primera etiqueta.

Declaración de un Array:

- private doublé[] operaciones;

Declaración y contrucción de objetos:

- Persona p = new Persona(“1”, “Alejandro”);

Los objetos se crean con el operador new.

Declaración de constructores:

- Tienen el nombre de la clase y no declaran tipo de retorno.
- Se permite sobrecarga.
- Si no se define un constructor, el compilador incluye el constructor por defecto.

El CLR de .NET incorpora un mecanismo de recolección de memoria dinámica: Garbage Collector.

Se puede declarar el método finalize() para liberar recursos que quedan fuera del entorno de ejecución.

Herencia

- Todos los objetos heredan directamente o indirectamente de object.
- La aplicación de métodos puede resolverse por ligadura estática o dinámica.
- El constructor de la clase hija tiene que invocar al de la clase padre utilizando la palabra clave base.

Polimorfismo

- El polimorfismo está permitido sólo para entidades de tipos referencia.

Ligadura

- La ligadura dinámica sólo se aplica en tipos referencia y en métodos declarados con el modificador virtual.
- La ligadura estática se aplica en el resto de los casos.

Clase Object

La clase object representa la raíz de la jerarquía de tipos en C# y .NET

Casting

Para los tipos referencia se usa el operador as.

- Estructurado = deposito as DepositoEstructurado;

Se puede usar is para consultar la compatibilidad de tipos:

- if(deposito is DepositoEstructurado){}

Clases abstractas

- Las clases pueden declararse como abstractas utilizando el modificador abstract.
- Los métodos y propiedades se pueden declarar abstractos con abstract.
- Si una subclase no implementa una declaración abstracta, debe declararse como abstracta.
- Una clase abstracta no puede ser instanciada.
- El objetivo de la clase abstracta es ser heredada.

Interfaces

- C# define el concepto de interfaz similar al de Java.
- Permite definir propiedades y métodos, pero no constantes.
- Una clase puede implementar múltiples interfaces.
- Una interfaz puede extender varias interfaces.
- Los miembros de una interfaz siempre son públicos.

Boxing: representación de tipos por valor como objetos por referencia.

Unboxing: realiza el proceso contrario.

Al igual que Java, podemos simular herencia múltiple utilizando interfaces y relaciones de clientela.

Genericidad

Una clase genérica puede ser parametrizada a cualquier tipo

Ejemplo:

```
Class Contenedor<T> {
    Public T Contenido {
        get; set;
    }
}
```

Estrategia

- Representación de una referencia a un método como un objeto.
- Las estrategias permiten establecer código como parámetro de un método.
- Para la definición de estrategias, C# declara el concepto delegado:
    - Similar a un puntero a función de C/C++.
    - Incluye control de tipos y permite definir referencias a métodos de instancia.

Ejemplo:

```
public delegate bool Test(Cuenta cuenta);
```

Excepciones

- La raíz de todas las excepciones es la clase System.Exception.
- Las excepciones predefinidas incluyen excepciones para el tratamiento de precondiciones:
    - ArgumentException: precondiciones de argumentos.
    - InvalidOperationException: precondiciones de estado.

Conclusiones

La programación orientada a objetos en C# es muy similar a la proporcionada por Java, muchos de estos conceptos se conservan, además algo de mi agrado es también la combinación con la programación orientada a objetos de C++, estas dos combinaciones es lo que para mí hace interesante a C#.

Referencias

http://dis.um.es/~bmoros/privado/apuntes/Curso09-10/POO6-CSharp-0910.pdf