Abstracción, claes y objetos

Ernesto Rodriguez

Universidad del Itsmo

erodriguez@unis.edu.gt

Abstracción

- Es un proceso que ocurre enteramente dentro de la mente del programador.
- Consiste en construir un objeto abstracto a partir de otro objeto.
- El objeto abstracto solamente contiene los detalles importantes según el contexto.
- El objeto abstracto ignora los demas detalles del objeto original.
- Existen varios (posiblemente infinitos) objetos abstractos que se pueden construir a partir de un objeto concreto.

Programación orientada a objetos

- Uno de los paradigmas principales de programación junto a *funcional* y *logica*.
- Agrupa attributos junto a operaciones sobre esos atributos en una estructura llamada objeto.
- Los objetos creados mediante abstracción pueden ser representados con objetos de la programación.
- El objeto tiene un estado, el cual es el valor actual de todos sus atributos.
- El estado del objeto se modifica mediante sus operaciones.

Clases

- Describen las propiedades y operaciones de un objeto.
- Se definen mediante un lenguaje de programación como C#.
- Los objetos abstractos pueden representarse mediante clases.
- Una clase permite crear uno o más objetos durante la ejecución de un programa.
- Un objeto creado a partir de una clase se llama *instancia de la clase*.

Instancia

- También se conocen como objetos.
- Se construyen a partir de una clase.
- Cada objeto existe en su propio espacio de memoria, incluso si fueron creados con la misma clase.
- Al asignar un objeto a una variable, la variable contiene una *referencia* al objeto.
- El estado del objeto se puede modificar mediante *metodos*.

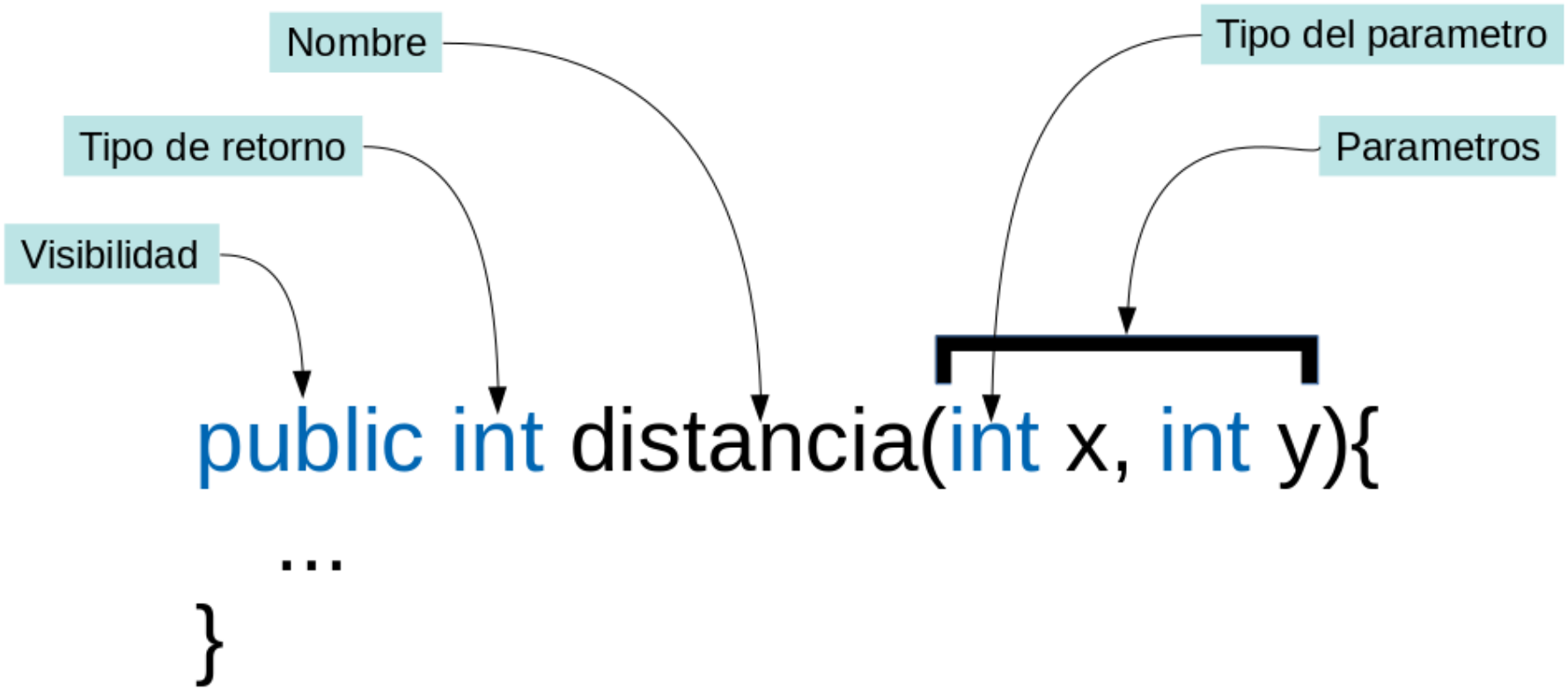
Propiedades

- Corresponden a valores almacenados en la memoria del objeto.
- Sirven como etiquetas que apuntan a una posición particular en memoria.
- Pueden ser *privadas* o *publicas*.
- Las propiedades privadas solo pueden ser accedidas por *metodos* de la clase instanciadora.
- Las propiedades publicas pueden ser accedidas por metodos ajenos a la clase instanciadora.
- Ejemplos: Longitud de un arreglo, posición del *Vehiculo*.

Métodos

- Los metodos son funciones asociadas a una clase.
- También pueden ser publicos o privados.
- El cuerpo de un metodo puede aceder a todas las propiedades y metodos de la clase instanciadora.
- Consequentemente, un método puede modificar el *estado* de un objeto.
- La palabra reservada *this* se utiliza para acceder propiedades y metodos del objeto al que le pertence el metodo.
- El metodo puede recibir parametros que pueda necesitar para realizar su trabajo.
- El metodo puede retornar un valor si lo desea.

Firma de un Método



Metodos puros y efectos secundarios

- Toda modificación al estado del objeto o del programa realizada por un metodo se llama *efecto.*
- Los efectos causados por un metodo pueden cambiar el comportamiento de otros metodos.
- Los metodos que no causan efectos y no son afectados por efectos de otros metodos se llaman *metodos puros*. Dichos, metodos son equivalentes a una función matematica.
- Los efectos dificultan poder rázonar sobre un programa y lo pueden volver impredecible.
- En el mundo ideal, un programa seria una función libre de efectos (ej. Haskell[1]), pero esto entra en conflicto con el proposito del programa, que es tener un efecto.
- Un metodo deberia tener **un** solo proposito y cumplir dicho proposito con la **menor** cantidad de efectos posibles.

Método constructor

- Metodo encargado de colocar a un objeto en su estado inicial.
- Solamente es llamado cuando un objeto nuevo es creado.
- Puede ser publico o privado.
- Se declara mediante el nombre de la clase.
- Se llama mediante la palabra reservada *new*.
- No retorna ningún valor.

Propiedades calculadas

- Un tipo especial de metodo disponible en C# y algunos otros lenguajes.
- La operación de lectura se define mediante la palabra `get`
- La operación de escritura se define mediante la palabra `set`.
- La operación de lectura **no** debe causar ningun efecto.
- La oparación de escritura **solamente** debe tener como efecto modificar la propiedad.
- Ejemplo: Distancia del origen.

Principio de única responsabilidad

- En general, los objetos representados por clases pueden ser complejos.
- Sin embargo, es importante que las clases tengan solamente **una** responsabilidad.
- Las clases complejas crean archivos largos y dificultan el razonamiento.
- Definir una clase a partir de clases más simples se conoce como *composición*.

Referencias


Haskell.org.
Haskell.
https://www.haskell.org/.