Verifica 14/05/2021 Ilaria Bonelli

1 Si può estendere il comportamento di un metodo per ritornare più valori in 3 modi: ritornando una classe/struct con tutti i valori necessari, usando le tuple che non sono altro che un insieme di oggetti (da c# 7.0 in poi) oppure usando la parola chiave out. Il più “performante” è il primo metodo, in cui vado ad incapsulare nelle classi Request e Response i dati che il metodo mi dovrà restituire. Tra le 3 opzioni, dunque, è meglio usare il request response pattern perchè la firma del mio metodo non la cambio mai, se voglio fare delle modifiche le faccio all’interno delle mie classi Request e Response, ma il resto del codice non lo devo toccare.

2 Il Casting è operazione di conversione tra i tipi. Esiste il casting esplicito: MyType x= (MyType) y; sarebbe comunque meglio avere un try catch perchè se per caso non riesce a fare il cast il codice esplode e mi genera un’eccezione. Il modo per gestire un casting senza usare le eccezioni è usare l’operatore “is” che serve a dirmi se un’istanza è di un certo tipo e quindi se posso fare la conversione: if (y is MyType) x=(MyType) y;. Oppure usando l’operatore “as” che sostanzialmente funge da try catch: prova a fare il cast, se riesce x contiene il valore convertito, se non riesce, x contiene il valore null: MyType x = y as MyType; if (x==null) Console.WriteLine(“Tipo non valido”). Naturalmente c’è anche il tipo implicito in cui non è richiesta alcuna sintassi speciale perchè la conversione ha sempre successo e non verranno persi dei dati: per esempio un intero posso assegnarlo ad una variabile di tipo long perchè sto portando un dato in un tipo che può contenerlo senza perdita di informazione e quindi lo posso fare senza problemi. Anche in una conversione da classe derivata a classe base non mi serve esplicitare che stiamo facendo un casting. Nelle conversioni esplicite dobbiamo indicare invece qual è l’operazione che stiamo facendo: lo uso quando ho per esempio un double e voglio portarlo dentro ad un intero; ovviamente ho perdita di informazione ma con questa consapevolezza posso fare un casting esplicito. Posso infine andare a definire i casting con i tipi dato creati da me: la sintassi è la stessa dell’operator preceduta da keyword implicit o explicit; il nome del metodo sarà il tipo in cui vogliamo convertire, mentre il parametro sarà il tipo da cui partiamo. Quando dobbiamo scrivere il casting tra due tipi possiamo realizzare la nostra operazione in uno dei due tipi coinvolti: l’overloading tra i due possiamo farlo o nel primo tipo o nel secondo (gestiti da c# in modo bidirezionale). “Is” e “As” però non sono supportati da questo tipo di overloading quindi poi non li posso usare (solo nella conversione esplicita).

3 La keyword static si può anteporre sia a una classe che ai membri. Una classe statica è una classe che non può essere istanziata. Per esempio, il Singleton pattern si fa in prima approssimazione con una classe static. I membri di una classe anche non static possono essere marcati static: posso istanziarla e utilizzare nella mia istanza ogni metodo che non sia statico (che si definiscono metodi d’istanza). Per richiamare ciò che è statico devo richiamare prima la classe: nomedellaclasse.nomedellaproprietà = valore;.

4) Un evento è un altro tipo di membro di una classe, insieme a campi, proprietà e metodi e serve ad attivare la comunicazione tra l’interno e l’esterno della mia classe. L’evento mantiene una lista di subscriber che vengono iterati per eseguire la notifica. Il publisher inoltra gli eventi a tutti i subscriber, ciascun subscriber deve essere aggiunto alla lista del publisher(subscribe) oppure rimosso(unsubscribe). La sottoscrizione usa l’overload di +=. Per sottoscrivere un evento si procede in questo modo: Istanza.Evento += f;. Per fare invece l’unsubscribe si fa l’overload di -= con la stessa sintassi vista sopra. Per fare tutto questo posso creare un delegato, dopodichè si crea l’evento di tipo “il mio delegato”, si solleva l’evento (sempre meglio che si abbia un controllo sul fatto che ci sarà almeno un subscriber prima di lanciarlo), si istanzia poi nel program in cui si genera il metodo che mi dà l’esito dell’evento. Oppure si sfrutta l’EventHandler già implementato da c# che è esso stesso un delegate; si realizza una classe che eredita da EventArgs in cui si inseriscono le proprietà e i metodi che vogliamo che il nostro evento utilizzi e poi si procede come sopra.

5) i generics sono interfacce, classi e metodi che prendono un tipo tra le parentesi angolari. È stato realizzato per scrivere del codice che andasse a lavorare in maniera indipendente dal tipo. Il generic ti consente di scrivere una sorta di template per definire come lavorare non con un singolo tipo ma con una serie di tipi differenti. L’implementazione è uguale ma non specifico il tipo. Si era pensato banalmente di sostituire tutti i tipi con “object” ma ciò non può funzionare: il linguaggio è fortemente tipizzato quindi perderei il controllo type safety. Col generic invece il compilatore si accorge del tipo che stiamo inserendo e quindi in sostanza generalizzo il codice ma senza perdere il type safety. Se voglio il default devo fare default(T), non faccio =0 e non posso assegnare null ad una classe generic. Possiamo usare i generics anche sui metodi, vale tutto ciò che vale per classi e interfacce.