Fondamenti di Programmazione

Antonia Sacchitella
Analyst Developer
Antonia.Sacchitella@icubed.it

"I bravi programmatori sanno cosa scrivere.
I migliori sanno cosa riscrivere."

ERIC STEVEN RAYMOND

Agenda

- Nelle puntate precedenti…
- Codice elegante
- Iterazione vs Ricorsione
- Sistemi di Version Control
  - Git e Azure DevOps
- Commentare il codice
- Flusso di dati da/verso file

icubed

Tipo

C# è un linguaggio detto «tipizzato»

Un tipo serve per descrivere il ruolo di una variabile nell'ambito del codice e ne caratterizza le funzionalità

Il linguaggio garantisce che il valore inserito nella variabile sia utilizzato in maniera coerente rispetto al tipo della variabile.

Value Type

- Contengono direttamente il dato nell'ambito dello stack del thread.
- Una copia di un Value Type implica la copia dei dati in esso contenuti.
- Le modifiche hanno effetto solo sull'istanza corrente.
- Contengono sempre un valore (null **non è** direttamente ammesso).
- I Value Type comprendono:
  - i tipi primitivi come int, byte, bool, ecc.
  - **enum**, **struct** (definiti dall'utente).

```
int i = 1;
bool b = true;
double d = 0.123;
DateTime a = DateTime.Now;
```

icubed

Reference Type

- Contengono solo un riferimento ad un oggetto nell'ambito dell'heap.
- La copia di un Reference Type implica la duplicazione del solo reference.
- Il reference che non referenzia nessuna istanza vale **null**.
- **Tutte le classi sono Reference Type!**

```
string s = "C#";
DataSet ds = New DataSet();
Person p = New Person();
```

icubed

Tipi di valore vs Tipi di riferimento

Tipo di valore	Tipo di riferimento
Contengono direttamente il valore dei dati	Contiene il riferimento alla locazione in memoria della variabile
Non devono essere instanziati	Deve essere instanziato
Non possono essere nulli	Può essere nullo

icubed

Routine

Insieme di istruzioni racchiuse in un unico blocco di codice che formano un'entità richiamabile in altri punti del codice.

L'elaborazione data dalla routine può avere:

- Parametri di input
- Valore di ritorno (risultato)

Le routine si suddividono in:

- **Funzioni** (hanno valore di ritorno)
- **Procedure** (non hanno valore di ritorno)

Dichiarazione Routine

- In sostanza la dichiarazione di un metodo è composta di:
  - zero o più keyword;
  - il tipo di ritorno del metodo oppure **void**;
  - il nome del metodo;
  - l'elenco dei parametri tra parentesi tonde.

- La *firma* (*signature*) di un metodo è rappresentata dal nome, dal numero dei parametri e dal loro tipo; il valore ritornato **non** fa parte della firma.

```
void MyMethod(string str) {
    // ...
}
```

```
int MyMethod(string str) {
    int a = int.Parse(str);
    return a;
}
```

icubed

Passaggio dei parametri

- I parametri di un metodo possono essere passati:
  - *By Value* (default)
  - *By Reference* (keyword **ref**)

```
int x = 0;
MyClass.MyMethod(2, 3, ref x);


public static void MyMethod(int a, int b, ref int c) {
    c = a + b;
}
```

icubed

Le regole del «codice elegante»

1. **Usare nomi descrittivi**
   - Le variabili sono i soggetti
   - I metodi sono le azioni

```
0 references
static void Main(string[] args)
{
    int q = 20;
    double b = 10.0;
    double a = Calc(q, b);
}

1 reference
private static double Calc(int q, double b)
{
    double s = 20;
    return (b - (b * s) / 100) * q;
}
```

```
0 references
static void Main(string[] args)
{
    int quantita = 20;
    double prezzoIniziale = 10.0;
    double prezzoScontato = CalcolaPrezzoFinale(quantita, prezzoIniziale);
}

1 reference
private static double CalcolaPrezzoFinale(int quantita, double prezzoIniziale)
{
    double sconto = 20;
    return (prezzoIniziale - (prezzoIniziale * sconto) / 100) * quantita;
}
```

icubed

Le regole del «codice elegante»

2. Dare a ogni componente uno scopo ben preciso

- Divisione in blocchi

```
1 reference
private static double CalcolaPrezzoFinale()
{
    Console.WriteLine("Inserisci quantità");
    Int32.TryParse(Console.ReadLine(), out int quantita);
    Console.WriteLine("Inserisci prezzo iniziale");
    double.TryParse(Console.ReadLine(), out double prezzoIniziale);
    Console.WriteLine("Inserisci sconto");
    double.TryParse(Console.ReadLine(), out double sconto);
    return (prezzoIniziale - (prezzoIniziale * sconto) / 100) * quantita;
}
```

```
0 references
static void Main(string[] args)
{
    int quantita = LeggiInfoIntero(); //Funzione lettura di un intero
    double prezzoIniziale = LeggiInfoDouble(); //Funzione lettura di un double
    double sconto = LeggiInfoDouble(); //Funzione lettura di un double
    double prezzoScontato = CalcolaPrezzoFinale(quantita, prezzoIniziale, sconto);
}

1 reference
private static double CalcolaPrezzoFinale(int quantita, double prezzoIniziale, double sconto)
{
    return (prezzoIniziale - (prezzoIniziale * sconto) / 100) * quantita;
}
```

icubed

Le regole del «codice elegante»

3. E’ riutilizzabile

Eliminare parti duplicate.

```
Console.WriteLine("Inserisci NOME: ");
string firstName = Console.ReadLine();
Console.WriteLine("Inserisci COGNOME: ");
string lastName = Console.ReadLine();
Console.WriteLine("Inserisci LUOGO DI NASCITA: ");
string placeBirth = Console.ReadLine();
```

```
0 references
static void Main(string[] args)
{
    string firstName = InserisciDato("NOME");
    string lastName = InserisciDato("COGNOME");
    string placeOfBirth = InserisciDato("LUOGO DI NASCITA");
}

3 references
private static string InserisciDato(string tipoDato)
{
    Console.WriteLine($"Inserisci {tipoDato}:");
    string dato = Console.ReadLine();
    return dato;
}
```

icubed

Le regole del «codice elegante»

4. Deve essere leggibile

Codice pulito != Codice intelligente

Anzi

Codice pulito > Codice intelligente

icubed

Le regole del «codice elegante»

5. E’ correttamente indentato

In termini prettamente visivi

```
int a = 10;
  if(a > 0)
{
      for(int i = 0; i < 10; i++){
          Console.WriteLine($"Sequenza di numeri {i}");
        }
}
  else
     {
      Console.WriteLine("Numero piccolo");
  }
}
```

```
int a = 10;
if (a > 0)
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine($"Sequenza di numeri {i}");
    }
}
else
{
    Console.WriteLine("Numero piccolo");
}
```

Visual Studio consente l’indentazione automatica

icubed

Le regole del «codice elegante»

6. Scegliere l’architettura giusta

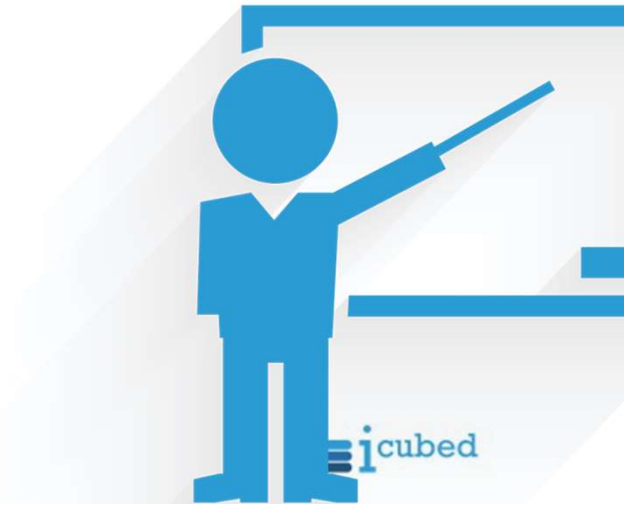
Dividere il codice in «file/classi» per concetti.

Consiglio pratico

Spesso è utile leggere e debuggare il codice “dei maestri”

Demo

Codice Elegante



icubed

Esercitazione n. 1

In una copisteria il costo unitario delle fotocopie varia a seconda del numero da effettuare secondo questa suddivisione:

- Tra 1-19 fotocopie 0,10€
- Tra 20-100 fotocopie 0,08€
- Maggiore di 100 fotocopie 0,05€

Inoltre se le fotocopie sono da rilegare viene aggiunto un costo di 1,80€.

Scrivere un programma che:

- prenda in input il numero di fotocopie da effettuare, il nome del cliente e se la rilegatura deve essere effettuata o meno

Esercitazione n. 1

Scrivere un programma che:

- Calcoli il prezzo totale
- Stampi il seguente prospetto:

  «Gentile Sig. ……
  il suo preventivo è di …… €
  compreso la rilegatura»

  N.B l'ultima riga è da scrivere solo se è richiesta la rilegatura.

Utilizzare le regole per scrivere un codice elegante.

icubed