# Metodi

Informatica
Prof. Viglietti Francesco
a.s. 2012-13

# Introduzione

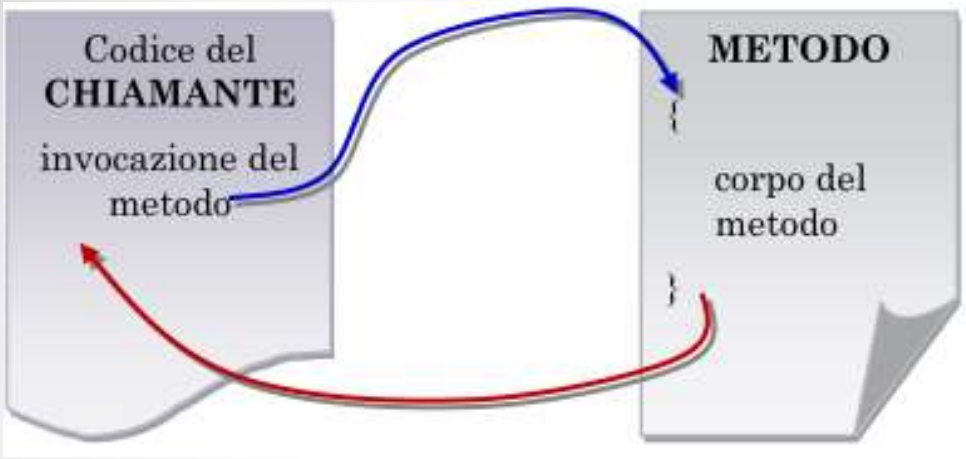
In C# parleremo di **metodi**, al posto di funzioni o di procedure. Il metodo è un oggetto che ha l'incarico di svolgere un compito e deve restituire un risultato.

Ad esempio: in una città le persone che vogliono ristrutturare un'abitazione devono presentare il progetto al comune, il quale risponde con un SI o NO. Il comune incarica un suo impiegato di svolgere questo compito.

In questa metafora:

- l'impiegato del comune è il **metodo**
- il cittadino che presenta richiesta edilizia è detto **chiamante**
- l'impiegato per poter svolgere il suo compito deve ricevere un progetto da esaminare; il progetto rappresenta un **parametro**!
- l'autorizzazione rappresenta il **risultato atteso** (ciò che il metodo restituisce al chiamante)
- quando un cittadino presenta la domanda all'impiegato sta chiamando il metodo **(invocazione)**
- quando il comune istruisce il suo impiegato sul compito da svolgere, è il momento di **definizione** del metodo

# Chiamata al Metodo




Il chiamante invoca il metodo, tramite il suo **nome**; il metodo svolge un lavoro e poi termina (con un risultato o meno); il controllo ritorna al chiamante che sfrutta l'esito del metodo e, se presente il risultato.

Usare i metodi offre diversi vantaggi :

1) Assegnare un nome a un compito specifico: raggruppando alcune istruzioni e attribuendo al gruppo un nome, che ne esprima lo scopo.
2) Riutilizzo: possibilità di richiamare il metodo più volte.
3) Pulizia del codice: se un programma è abbastanza grosso, conviene suddividere il programma in frammenti, ciascuno individuato da un metodo.

# Definizione del metodo

La **dichiarazione**, stabilisce il comportamento del metodo. La sintassi generale di un metodo in C# è la seguente:

***//dichiarazione di un metodo in C#***

***<descrittore> <tipo> NomeMetodo (elenco_parametri*** *Opzionale* ***)***

***{ //corpo del metodo***

***<corpo>***

***}***

Descrittore

Indica come lavora il metodo. Esamineremo 3 possibili Descrittori.

Static: si usa per i metodi in modalità Console, lo ometteremo in Visuale.

Public / Private: è necessario usarli SEMPRE.

Tipo

Determina se il metodo rende un dato oppure no. Se non rende alcun dato, si deve indicare void. Se invece rende un dato allora occorre indicare il tipo del dato atteso.

Nome

Determina come si chiama il metodo, e serve anche per invocarlo in seguito.

Lista parametri

Un metodo può non avere parametri, o averne uno, o averne più di uno. I parametri servono per scambiare dati con l'esterno, e vanno separati da virgole.

# Esempi

```
static public int Rende18() //metodo senza parametri
{    return 18;  }

static public int Quadrato(int x) // metodo con 1 parametro
{    return x*x;   }

static public int Massimo(int x , int y) // metodo con 2 parametri
{
  if (x > y)
    return x;
  else  return y;
}

static public void Scambia(ref int x, ref int y) // metodo con 2 parametri
{
  int t = x;
  x= y;   y = t;
}
```

# Invocazione

L'invocazione stabilisce l'uso del metodo. Questo comporta una deviazione del flusso di elaborazione e costringe il programma ad eseguire il corpo del metodo. In questa fase accadono i seguenti fatti:

1. il codice del chiamante viene temporaneamente interrotto
2. gli argomenti indicati nella chiamata vengono passati ai parametri definiti con il metodo
3. viene eseguito il corpo del metodo
4. quando il codice del corpo termina OPPURE si incontra il ***return*** il controllo ritorna al chiamante
5. il codice del chiamante riceve il risultato del metodo e può proseguire

# Esercizio guidato (defin.)

1 Prepara un progetto con un form simile alla Fig.1.


2 apri il codice dell'applicazione. Posiziona il cursore di scrittura prima del costruttore Form1 e scrivi:

```
public partial class Form1 : Form
{
    //dichiarazioni dei metodi
    public void ColoraFinestra()
    {
        this.BackColor = Color.Red;
    }

    public Form1()
    {
        InitializeComponent();
    }
}
```

Dichiarazione di metodo

Corpo del metodo

Intestazione

Costruttore

3 Continua a dichiarare un secondo, un terzo e un quarto metodo; sotto il primo…

```
public void MostraNelTitolo (string
frase)
{
    this.Text = frase;
}
```

```
public string RestituisciNome()
{
    Random estrai = new Random();
    if (estrai.Next(1, 3) == 1)
        return "Ares";
    if (estrai.Next(1, 3) == 2)
        return "Ephest";
    if (estrai.Next(1, 3) == 3)
        return "Aphrodith";
    return "Impossibile";
}
```

```
public double CalcolaMedia (int numero, int altroNumero)
{
    double media;
    media = (numero + altroNumero) / 2.0;
    return media;
}
```

# Esercizio guidato (invoc.)

Ora invocheremo i metodi dichiarati prima. Per invocare un metodo è necessario utilizzare il suo nome (identificatore) le parentesi tonde e, al loro interno, indicare gli argomenti compatibili in numero e tipo.

Associa al primo pulsante il codice seguente:

```
ColoraFinestra();//invocazione del metodo
```

Associa al secondo pulsante il codice seguente:

```
string s;
s = RestituisciNome(); //invocazione del metodo
MessageBox.Show (s);
```

Associa al terzo pulsante il codice seguente:

```
string s;
s = "Ottimo Motto ";
MostraNelTitolo (s);
```

Associa al quarto pulsante il codice seguente:

```
int x = 13;
int y = 17;
double media;
media = CalcolaMedia (x, y)
MessageBox.Show ("risultato: " + media);
```

# Metodi senza parametri 1

***Metodi senza risultato né parametri***

Un metodo senza risultato non rende niente, ma non vuol dire non fa niente! Ipotizziamo un'azienda che assuma dei ragazzi per distribuire volantini; i ragazzi prendono i volantini e vanno in giro a consegnarli, ma al ritornano in azienda non riportano nulla; in pratica hanno svolto un lavoro, anche se non hanno nulla da riconsegnare.

Un metodo senza risultato si dice di tipo void (vuoto) equivale a nessun valore.

Un metodo senza parametri, lavora senza informazioni dal chiamante; e dopo il nome del metodo, tra le parentesi tonde non c'è nulla (ma le parentesi sono obbligatorie).

***Dichiarazione ed invocazione***

Vediamo il caso di alcuni metodi senza risultato né parametri:

```
//Dichiarazione  metodo senza risultato né parametri
public void ColoraFinestra()
{
  if (this.BackColor == Color.Red)
    this.BackColor = Color.Blue;
  else
    this.BackColor = Color.Red;
}
```

```
//Invocazione  metodo senza risultato né parametri
//puoi invocarlo da un pulsante
button1
ColoraFinestra();
```

# Metodi senza parametri 2

1. Nella dichiarazione compare **void**, quindi il metodo non rende valori e può essere invocato come se fosse un comando.
2. Il nome del metodo è obbligatorio, e lo individua senza ambiguità. Quando si invoca un  metodo si usa il suo nome.
3. Tra le parentesi non c'è nulla, ovvero senza parametri. Le parentesi sono sempre obbligatorie sia in fase dichiarativa che invocativa. Se un metodo è dichiarato senza parametri (con parentesi vuote) DEVE essere invocato senza parametri (con parentesi vuote).

***Metodi con risultato e senza parametri***

Un metodo con risultato deve rendere un valore; che sarà restituito al codice che effettua l'invocazione (chiamante).

Ad esempio… ipotizziamo che un docente chieda ad un alunno di fare una colletta per una buona opera; l'alunno va in giro, raccoglie i soldi ma infine deve tornare dal docente per consegnarli a lui; in pratica ha svolto un lavoro che rende un frutto, e questo esito va reso a colui che ha richiesto il lavoro. Una  metodo con risultato deve specificare il tipo del valore restituito.

# Metodi senza parametri 3

**Dichiarazione ed invocazione**

Vediamo il caso di alcuni metodi con risultato ma senza parametri:

```
//dichiarazione  metodo con risultato senza parametri
public int EstraiNumeroPari()
{
  Random estrai = new Random();
  return estrai.Next(1,10)*2;
}
```

```
//Invocazione  metodo con risultato senza parametri
//puoi invocarlo da un pulsante
button1
int numero;
numero = EstraiNumeroPari();
```

Nella dichiarazione, **int** corrisponde al valore da rendere. Quando si invoca metodo, usando il suo nome, è possibile usare il valore che renderà, per es. per assegnarlo ad una variabile. Nell'es. precedente il chiamante dichiara una variabile (numero) in cui sarà depositato il valore reso dal metodo. Un metodo con un tipo, deve avere un **return** dentro il corpo per assicurare che renda un valore.

# Metodi senza parametri 4

Esempio di metodo di tipo Array

```
//Dichiarazione  metodo con risultato senza
parametri
public int[] CreaVettore()
{
  int[] vettore = new int[100];
  Random estrai = new Random();
  for (int i=0; i<100; i++)
    vettore [i] = estrai.Next(10,100);
  return vettore;
}
```

```
//Invocazione  metodo con risultato senza
parametri
//puoi invocarlo da un pulsante
button1
int[] risultato;
risultato = CreaVettore();
```

Nella dichiarazione, il tipo restituito è un vettore di interi; anche in questo caso, si può osservare che il tipo restituito deve coincidere col tipo che segue la parola chiave return.

# Esempi

Predisporre un form con 3 pulsanti e 3 caselle di testo; si noti che uno stesso pulsante può invocare più di un metodo; poi si risolvano i seguenti esercizi:

1) Dichiarare un metodo che cambi il titolo della finestra aggiungendo una "a" alla fine; il pulsante 1 l'invoca 1 volta, il pulsante 2 l'invoca 2 volte

2) Dichiarare un metodo che restituisca un colore casuale scelto tra tre possibili; il pulsante 1 l'invoca 1 volta

3) Dichiarare un metodo che scambia il testo della textBox1 con la textBox2; il pulsante 2 l'invoca una volta

4) Dichiarare un metodo che sposta casualmente di 10px in orizzontale il form1; il pulsante 3 l'invoca una volta

5) Dichiarare un metodo che sposta casualmente di 10px in verticale il form1; il pulsante 3 l'invoca una volta

6) Dichiarare un metodo che decide casualmente se invocare il metodo di spostamento orizzontale o quello di spostamento verticale; il pulsante 2 l'invoca una volta

7) Dichiarare un metodo che restituisca un valore con la virgola compreso tra 0 e 10 (per es. 1,92 oppure 9,001); il pulsante 1 l'invoca una volta

# Metodi con parametri

I parametri sono un modo con cui è possibile far comunicare il metodo con il chiamante. I parametri sono delle locazioni (come delle variabili), che esistono solo all'interno del metodo, in cui ospitare i dati che saranno oggetto della comunicazione tra metodo e chiamante. I parametri vanno dichiarati in fase dichiarativa. Quando si invoca il metodo è necessario utilizzare tanti argomenti quanti sono i parametri dichiarati, e gli argomenti devono essere dello stesso tipo.

**Metodi senza risultato coi parametri**

Come già visto, un metodo senza risultato è di tipo void.

Un metodo con i parametri lavora con informazioni passate dal chiamante; quando un metodo è con parametri significa che dopo il nome, tra le parentesi tonde sono elencati i parametri. Questi nel passaggio per valore sono elencati separati da virgole, ciascuno nella forma <tipo nome>: in pratica è come dichiarare delle variabili (tra le parentesi) senza assegnare loro alcun valore iniziale.

# Esempi

```
//dichiarazione funzione senza risultato con parametri
public void Colora (int x, int y)
{
  if (x > y)
    this.BackColor = Color.Blue;
  else
    this.BackColor = Color.Red;
}
```

```
//invocazione funzione senza risultato con parametri
//puoi invocarla da un pulsante button1
Colora (13 , 17);

int a =  -17;
Colora (a, -11);
```

Nella dichiarazione compare void, perciò il metodo è senza risultato.

Il nome del metodo dovrebbe esemplificare il suo scopo.

Nella definizione, tra le parentesi ci sono più parametri, ciascuno col suo tipo; nell'invocazione ci sono argomenti in numero e tipo coerenti con la definizione.

# Metodi con risultato e parametri

Quando un metodo è definito con risultato e con parametri, significa che il metodo deve avere dati dal chiamante per poter lavorare, e alla fine della sua elaborazione restituisce al chiamante un valore del tipo specificato.

Vediamo il caso di alcuni metodi con risultato e con dei parametri:

```
//dichiarazione funzione con risultato e parametri
public double Media (int a, int b)
{
  return ((a + b)/2.0);
}
//invocazione funzione con risultato e parametri
double m =  0.0;
m = Media (13 , 17);
```

Nella dichiarazione compare double, cioè il metodo dovrebbe essere invocato per utilizzare il risultato, per esempio in una assegnazione o in una espressione.

Il nome del metodo è obbligatorio.

Tra parentesi sono elencati i parametri; quando si scrive un'invocazione si devono elencare gli argomenti nello stesso numero dei parametri e devono avere tipo compatibile.

# I parametri

I parametri costituiscono il principale strumento di comunicazione fra un metodo e il chiamante. Questa comunicazione può avvenire in diversi modi, con differenti significati. In Visual C# esistono tre diversi modi di passare i parametri:

- Passaggio per valore
- Passaggio per riferimento
- Passaggio per risultato

Ciascuno di questi modi dev'essere usato quando è opportuno facendo attenzione alle differenze che li contraddistinguono.

In un metodo con più parametri ciascuno di questi deve indicare il suo specifico modo di passaggio.

La lista dei parametri quindi appare con la seguente sintassi:

**Lista parametri = <modo><tipo><nome>,<modo><tipo><nome> , . . . , <modo><tipo><nome>**

# Passaggio per valore

Se non si indica alcun modo di passaggio, il compilatore lo considera un passaggio per valore. Nel passaggio per valore il parametro deve indicare sempre il tipo seguito dal nome. Il passaggio per valore si scrive nel seguente modo:

**parametro = <tipo><nome>**

Es. Consideriamo un metodo che debba restituire il valore più piccolo tra 2 valori interi ricevuti in ingresso; allora il metodo si può schematizzare come segue:


```
public int Minimo (int x , int y)
{
  if (x < y)
    return x;
  else
    return y;
}
```

- Il tipo restituito è un intero
- Il nome è obbligatorio e dovrebbe esprimere l'obiettivo che si propone.
- I parametri sono entrambi passati per valore; ciascuno deve indicare il tipo e il nome.

# Passaggio per valore 2

```
//esempio invocazione 1
int min;
min = Minimo ( 13, -17);
Text = "" + min ;
```

```
//esempio invocazione 2
int a = -13, b = 7;
int ris = Minimo ( a, b);
Text = "" + ris ;
```

Sopra due diverse modalità di invocazione del metodo Minimo().

Il passaggio per valore comporta la copia del valore degli argomenti nei parametri, ovvero, si devono valutare gli argomenti e stabilire il loro valore; questo viene copiato negli identificatori dei parametri, che tuttavia restano distinti dagli argomenti.

Nel caso dell'invocazione **1**, il metodo Minimo è invocato con 2 argomenti, i valori 13 e -17, sono copiati nei parametri x ed y; quindi prima di iniziare il metodo x vale 13 e y vale -17; quando inizia… il metodo controlla se (x<y) il cui valore è false; il metodo esegue il ramo else che impone al metodo di terminare col risultato -17 che è reso al chiamante e lo copia nella variabile min.

Nell'invocazione **2**, il metodo Minimo è invocato con 2 argomenti: le variabili a e b. Quindi si procede prima a valutare la a (-13) e la b (7). Il valore -13 è copiato nel parametro x, mentre il valore 7 è copiato nel parametro y. Adesso gli spazi di memoria di a e b sono del tutto distinti da quelli di x e y. Se la funzione modifica x oppure y, non ci sarà alcun effetto né su a né su b. Il metodo lavorerà come ci si aspetta, e rende -13.

# Passaggio per riferimento 1

```
public void Scambio (int x , int y)
{
  int tmp = x;     x = y;   y = tmp;
}
```

```
int a = 13, b = 17;
Scambio ( a , b );
MessageBox.Show ("a: " + a + " b: " + b);
Scambio ( a , 1 );
MessageBox.Show ("esito: " + a);
```

Consideriamo il metodo Scambio, che si propone di scambiare i valori contenuti in due variabili. Analizziamo cosa accade quando si invoca la funzione con gli argomenti a e b: i loro rispettivi valori sono13 e 17 che sono copiati nei parametri x e y; quindi x vale 13 e y vale 17.

Adesso il metodo scambia i valori contenuti in x e y, infine termina. Ma quando si prova a visualizzare i valori contenuti in a e b si riscontra che i valori NON sono stati scambiati! Purtroppo è stato usato il passaggio per valore, che mantiene separati gli spazi dei parametri da quelli degli argomenti; quindi quando si modificano i parametri, non si ottiene alcun effetto sugli argomenti.

# Passaggio per riferimento 2

Il passaggio per riferimento, consente di condividere lo spazio degli argomenti con quello dei parametri; deve essere reso esplicito, scrivendo **ref** prima della dichiarazione del parametro rispettivo:

**parametro = ref <tipo><nome>**

Vediamo ancora il metodo dello scambio, ma utilizzando il passaggio per riferimento:

```
public void Scambio (ref int x , ref int y)
{
  int tmp = x;     x = y;   y = tmp;
}
```

- Il tipo restituito è un intero.
- Il nome è obbligatorio e dovrebbe esprimere l'obiettivo che si propone.
- I parametri sono entrambi passati per riferimento; e ciascuno deve indicare il tipo e il nome.

# Passaggio per riferimento 3

```
//esempio invocazione 1
int a = 13, b = 17;
int Scambio (ref a, ref b);
Text = "" + a;
```

```
//esempio invocazione 2
int a = 3;
int Scambio ( ref a, 17);
Text = ''ERRORE!!!!'' ;
```

```
//esempio invocazione 3
int a = -13;
int Scambio ( ref a, ref a);
Text = «Errore??» ;
```

Sopra 3 diverse modalità di invocazione del metodo Scambio(). L'invocazione **1** è OK. N.B. è necessario indicare sempre ref anche nell'invocazione!

L'invocazione **2** solleva un errore di sintassi: una costante non può essere passata per riferimento, poiché non può essere mai modificata!

L'invocazione **3** potrebbe NON procedere, nel caso in cui l'argomento a non sia inizializzato; in pratica prima di usare la variabile come argomento deve avere un valore già specificato!

Nel passaggio per riferimento viene condivisa la memoria dell'argomento con quella del parametro. In pratica, se un metodo compie una qualsiasi modifica su un parametro passato per riferimento, si modifica il valore dell'argomento. Il passaggio per riferimento si usa quindi ogni volta che si desidera che gli argomenti riflettano le modifiche operate sui parametri.

Nel passaggio per riferimento è necessario che:

- L'argomento abbia un valore prima di essere invocato;
- L'argomento sia una locazione modificabile (non una costante);

# Passaggio per risultato 1

```
public void NomeCasuale (ref string s)
{
  Random dado = new Random();
  int valore = dado.Next (1,7);
  switch (valore)
  case 1: return "Abele";
  case 2: return "Bruto";
  case 3: return "Caio";
  case 4: return "Dante";
  case 5: return "Era";
  case 6: return "Fedro";
}
```

```
int nome;
NomeCasuale (ref nome);
MessageBox.Show ("esito: " + nome);
int altroNome;
altroNome = "Giuda";
NomeCasuale (ref altroNome);
MessageBox.Show ("esito: " + altroNome);
```

Nell'invocazione con l'argomento nome, si solleva un errore, poiché il parametro richiede che vi sia già un valore specificato per l'argomento.

Nella seconda invocazione, con l'argomento altroNome, si risolve il problema indicando un valore iniziale (Giuda), che però non serve a nulla, salvo che a evitare l'errore. Questo nome è inutile e può creare dubbi in chi dovesse leggere e modificare il programma.

# Passaggio per risultato 2

Il passaggio per risultato, consente di condividere lo spazio degli argomenti con quello dei parametri, deve essere reso esplicito, scrivendo **out** prima della dichiarazione del parametro rispettivo:

parametro = **out** <tipo><nome>

```
public void NomeCasuale (out string s)
{
  Random dado = new Random();
  int valore = dado.Next (1,7);
  switch (valore)
  case 1: return "Abele";
  case 2: return "Bruto";
  case 3: return "Caio";
  case 4: return "Dante";
  case 5: return "Era";
  case 6: return "Fedro";
}
```

# Passaggio per risultato 3

```
//esempio invocazione 1
string nome;
NomeCasuale (out nome);
MessageBox.Show (nome);
```

```
//esempio invocazione 2
string altroNome = "Giuda";
NomeCasuale (out altroNome);
MessageBox.Show (altroNome);
```

```
//esempio invocazione 3
NomeCasuale (out "Giuda");
```

Sopra 3 diverse modalità di invocazione del metodo Nomecasuale(). L'invocazione **1** è OK. N.B. è necessario indicare sempre out anche nell'invocazione!

L'invocazione **2** è OK: il nome "Giuda" è inutile e verrà sostituito.

L'invocazione **3** solleva un errore poiché è indispensabile che l'argomento sia una locazione, per esempio una variabile!

Il passaggio per risultato è simile a quello per riferimento. A differenza del riferimento nel corpo del metodo è necessario che vi sia una assegnazione al parametro, per evitare il rischio che il metodo si concluda col parametro senza alcun valore. Un'altra differenza è che nell'invocazione non è necessario specificare un valore iniziale: infatti spetta al metodo inizializzare l'argomento, proprio forzando l'assegnazione nel suo corpo.

# Riepilogo

Quando si invoca un metodo occorre tenere ben distinti i parametri formali (indicati nella definizione del metodo) dagli argomenti attuali (specificati al momento dell'invocazione). I parametri formali ESISTONO SOLO nell'ambiente locale del metodo, e fuori da esso NON ESISTONO.

Gli argomenti attuali della chiamata esistono solo se sono visibili nell'ambiente del chiamante, ovvero sono utilizzabili dal frammento di codice che sta invocando il metodo.

Quando il metodo viene chiamato, il sistema alloca memoria per poter eseguirlo. In memoria si sistemano spazi adatti sia ai parametri formali che alle variabili locali alla funzione. I parametri sono locali alla funzione che li usa come dei segnaposto per svolgere ipotetici calcoli.

**Passaggio per valore**

Non si specifica alcun modo nel parametro. I valori degli argomenti sono copiati nei parametri, che lavorano su copie dei valori, non sugli originali. Per questo eventuali modifiche sui parametri NON HANNO EFFETTO sugli argomenti.

**Passaggio per riferimento**

Se si desidera che gli argomenti passati possano essere modificati dal metodo, utilizzando la parola ref, e specificando ref anche per l'argomento.

Questo passaggio implica che il metodo possa modificare l'argomento, ne consegue che l'argomento non può essere una costante e deve essere inizializzato prima di invocarlo.

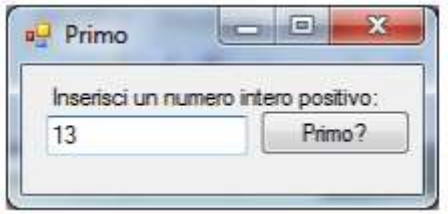
**Passaggio per risultato**

Se si desidera che gli argomenti possano non essere inizializzati e che spetti al metodo attribuire loro un valore. Si dichiara utilizzando la parola out, e si specifica out anche nell'invocazione. I parametri lavorano proprio sugli argomenti, non su copie.

# Esercizi su metodi

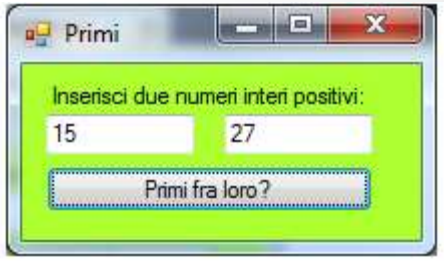
**Esercizio 1. Numero primo**

Definire un metodo che restituisce un valore booleano che indica se un numero passato è primo. L'utente scrive un numero nella textBox; il pulsante invoca il metodo e poi visualizza un messaggio con l'esito restituito.



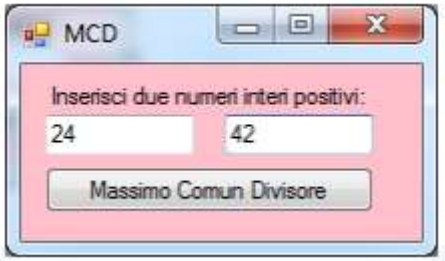

**Esercizio 2. Numeri primi fra loro**

Definire un metodo che restituisce un valore booleano che indica se due numeri sono primi fra loro (hanno solo 1 come divisore comune). L'utente scrive nelle due textBox dei numeri; il pulsante invoca il metodo e poi visualizza un messaggio con l'esito restituito.




**Esercizio 3. MCD**

Definire un metodo che calcola il massimo comun divisore tra due numeri interi positivi. L'utente scrive nelle due textBox dei numeri; il pulsante invoca la funzione e poi visualizza un messaggio con l'esito restituito.




**Esercizio 4. mcm**

Definire un metodo che calcola il minimo comune multiplo tra due numeri interi positivi. L'utente scrive nelle due textBox dei numeri; il pulsante invoca il metodo e poi visualizza un messaggio con l'esito.