BCSCHOOL

Empowering Digital Skills For The Jobs Of The Future

by

BC Soft
consulenza sistemi informativi

AiRED
Artificial Intelligence Ricerca e Sviluppo

Academy .NET

Classi e Oggetti

Sommario

- Classi e oggetti
  - Cosa sono gli oggetti?
  - Cosa sono le classi?
- Classi in C #
  - Dichiarazione di classe
  - Campi e proprietà: istanza e static
  - Istanza e metodi statici
  - Costruttori
- Enumerazioni
- Strutture
- Tipi parziali
- Tipi anonimi

Cos’è un oggetto?

Gli oggetti in un software modellano oggetti del mondo reale o concetti astratti.

Esempi:

banca, conto, cliente, cane, bicicletta, coda

Gli oggetti del mondo reale hanno stati e comportamenti.

Stati dell'account:

titolare, bilancio, tipo

Comportamenti dell'account:

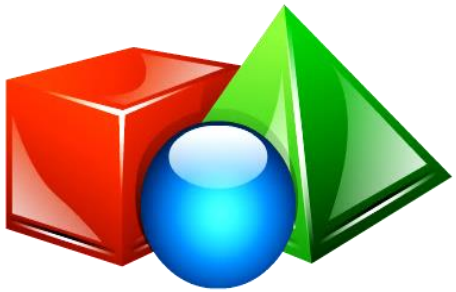
ritirare, depositare, sospendere

Cos'è un oggetto?

In che modo gli oggetti software implementano oggetti del mondo reale?

- Usano **variabili** / dati per implementare gli stati
- Usano **metodi** / funzioni per implementare comportamenti

Un oggetto è un pacchetto software di variabili e metodi correlati.



Rappresentazione degli oggetti

- conto
- persona
- cliente

oggetti del mondo reale

……

- numeri
- caratteri
- queues
- arrays

oggetti astratti

Definizione di Classe

Le **classi** agiscono come modelli da cui viene creata un'istanza di un oggetto in fase di esecuzione.

Le classi definiscono le proprietà dell'oggetto e i metodi utilizzati per controllare il comportamento dell'oggetto.

La Classe

La **classe** fornisce **la struttura per gli oggetti**

Definisce il loro prototipo, funge da modello

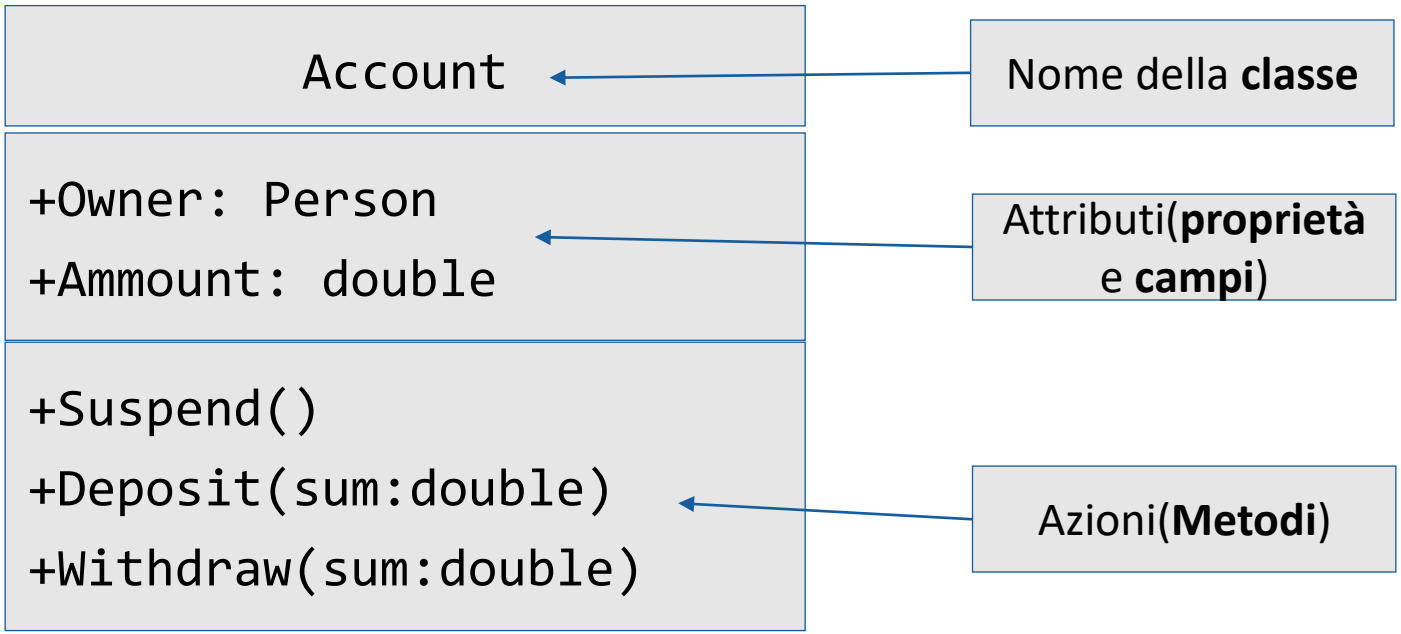
Le classi definiscono:

Un insieme di **attributi** rappresentato da campi e proprietà

Insieme di **azioni** rappresentato dai metodi

Una classe definisce i metodi e i tipi di dati associati a un oggetto

La Classe


Account
+Owner: Person
+Ammount: double
+Suspend()
+Deposit(sum:double)
+Withdraw(sum:double)
Nome della **classe**
Attributi(**proprietà** e **campi**)
Azioni(**Metodi**)

Gli Oggetti

Le classi forniscono **la struttura** per gli oggetti.

Definisce il loro prototipo, funge da modello.

Un oggetto è un'istanza concreta di una particolare classe

La creazione di un oggetto da una classe è chiamata **istanziazione**

Gli oggetti hanno uno **stato:**

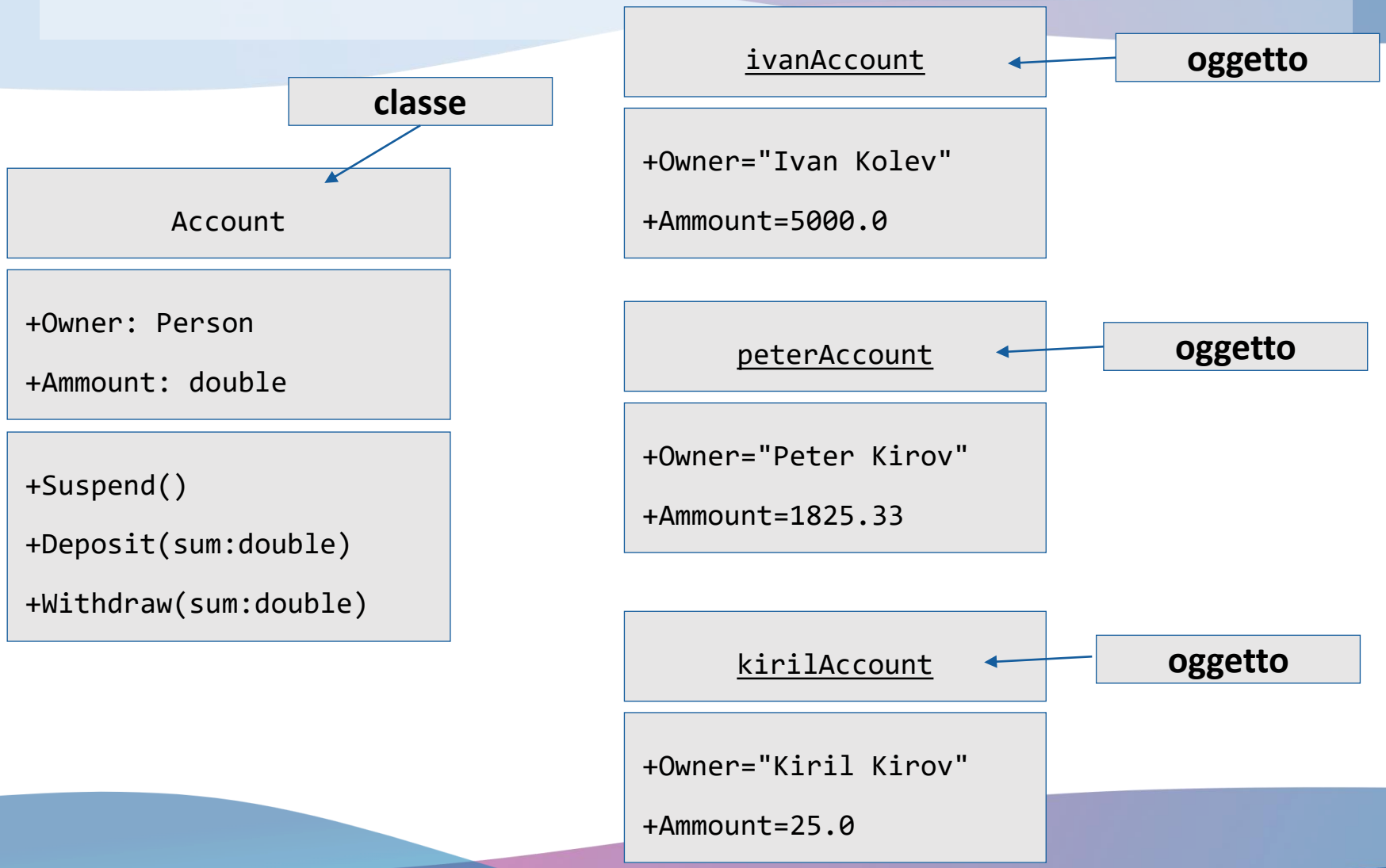
Un insieme di valori associati ai loro attributi

Gli Oggetti

Esempio:

- Classe: **Account**
- Oggetti: **account di Ivan**, **account di Peter**

Gli oggetti


classe
Account
+Owner: Person
+Ammount: double
+Suspend()
+Deposit(sum:double)
+Withdraw(sum:double)
ivanAccount
oggetto
+Owner="Ivan Kolev"
+Ammount=5000.0
peterAccount
oggetto
+Owner="Peter Kirov"
+Ammount=1825.33
kirilAccount
oggetto
+Owner="Kiril Kirov"
+Ammount=25.0

Classi in C#

Classi - unità di base che compongono i programmi.

L'implementazione è **incapsulata** (nascosta).

Le classi in C # possono contenere:

- **Campi** (variabili membro)
- **Proprietà**
- **Metodi**
- **Costruttori**
- **Inner Types**
- Ecc. (**Eventi**, **indicizzatori**,...)

Classi in C# - Esempio

Esempi di classi:

- **System.Console**
- **System.String (string in C#)**
- **System.Int32 (int in C#)**
- **System.Array**
- **System.Math**
- **System.Random**
- **System.DateTime**
- **System.Collections.Generics.List<T>**

Definizione di una classe

Le classi semplici contengono alcuni campi di dati, ad es.

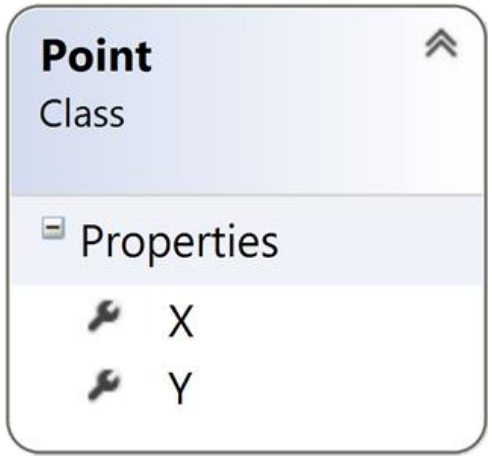
```
class Point
{
  public int X { get; set; }
  public int Y { get; set; }
}

Point p = new Point() { X = 5, Y = 7 };
Console.WriteLine("Point({0}, {1})", p.X, p.Y);
```

Nome Classe

proprietà

Creazione di un oggetto

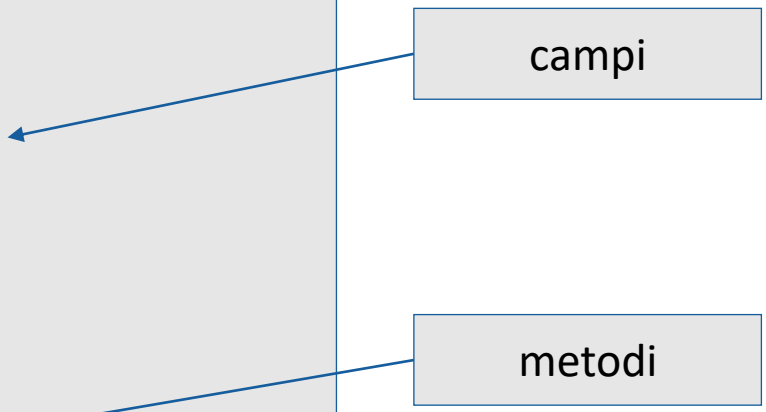


Classi - Struttura

Le classi possono definire **dati** (campi) e **operazioni** (metodi)

```
class Rectangle
{
  public int Top { get; set; }
  public int Left { get; set; }
  public int Width { get; set; }
  public int Height { get; set; }

  int CalcArea()
  {
    return Width * Height;
  }
```



Classi - Struttura

proprietà

```
public int Bottom
{
  get
  {
    return Top + Height;
  }
}
```

proprietà

```
public int Right
{
  get
  {
    return Left + Width;
  }
}
```

metodo booleano

```
public bool IsInside(Rectangle r)
{
   return (r.Left <= Left) && (r.Right >= Right) &&
      (r.Top <= Top) && (r.Bottom >= Bottom);
}
```

Dichiarare oggetti

Un'istanza di una classe può essere definita come qualsiasi altra variabile:

```
using System;
...
// Define two variables of type DateTime
DateTime today;
DateTime halloween;
```

Le istanze possono essere utilizzate per inizializzare

```
// Declare and initialize a structure instance
DateTime today = DateTime.Now;
```

Campi e proprietà

I campi sono membri (data members) di una classe

Possono essere variabili e costanti (sola lettura)

L'accesso a un campo non richiama alcuna azione dell'oggetto

Accede solo al suo valore

Esempio:

String.Empty (stringa **""**)

Accesso ai campi

I campi **costanti** sono di sola lettura.

I campi variabili possono essere letti e modificati.

Di solito le **proprietà** vengono utilizzate per evitare l'accesso diretto ai campi:

```
// Accessing read-only field
String empty = String.Empty;

// Accessing constant field
int maxInt = Int32.MaxValue;
```

Le Proprietà

Le proprietà assomigliano ai campi

- Hanno nome e tipo
- Possono contenere codice eseguito all'accesso

Solitamente utilizzate come **wrapper**

- Per controllare l'accesso ai campi
- Possono contenere una logica più complessa

Comprendono due componenti denominati **getters** e **setters**

- **get** per leggere il valore di un campo
- **set** per modificare il valore di un campo

Le Proprietà

A seconda delle funzioni di accesso implementate, le proprietà possono essere:

- Sola lettura (**get**)
- Lettura e scrittura (**get** e **set**)
- Sola scrittura (**set**)

Esempio di proprietà di sola lettura:

String.Length

Esempio di proprietà di lettura / scrittura:

Console.BackgroundColor

Accesso a proprietà e campi - Esempio

```
using System;
...
DateTime christmas = new DateTime(2009, 12, 25);
int day = christmas.Day;
int month = christmas.Month;
int year = christmas.Year;
Console.WriteLine(
  "Christmas day: {0}, month: {1}, year: {2}",
  day, month, year);
Console.WriteLine(
  "Day of year: {0}", christmas.DayOfYear);
Console.WriteLine("Is {0} leap year: {1}",
  year, DateTime.IsLeapYear(year));
```

Istanze e membri statici

Campi, proprietà e metodi possono essere:

Di istanza (o membri oggetto)

Statici (o membri della classe)

I **membri di istanza** sono specifici per ogni oggetto

Esempio: persone diverse hanno nomi diversi

I **membri statici** sono comuni a tutte le istanze di una classe

Esempio: **DateTime.MinValue** è condiviso tra tutte le istanze di **DateTime**

Accesso ai membri

Accesso ai membri dell'istanza

Il nome dell'istanza, seguito dal nome del membro (campo o proprietà), separato da un punto (".")

```
<instance_name>.<member_name>
```

Accesso ai membri statici

Il nome della classe, seguito dal nome del membro

```
<class_name>.<member_name>
```

Istanze e membri statici - Esempi

Esempio di membro di istanza

String.Length

Ogni oggetto stringa ha una lunghezza diversa

Per esempio. "I like C#" Lunghezza -> 9

Esempio di membro statico

Console.ReadLine()

La console è una sola (globale per il programma)

La lettura dalla console non richiede la creazione di un'istanza

Metodi

I metodi manipolano i dati dell'oggetto a cui appartengono o eseguono altre attività.

Esempi:

Console.WriteLine(...)
Console.ReadLine()
String.Substring (index, length)
Array.GetLength (index)
List <T> .Add (element)
DateTime.AddDays (count)

Metodi di istanza

I **metodi di istanza** manipolano i dati di un oggetto specificato o eseguono altre attività

- Se viene restituito un valore, dipende dalla particolare istanza della classe

Sintassi:

- Il nome dell'istanza, seguito dal nome del metodo, separato da un punto

```
<object_name>.<method_name>(<parameters>)
```

Chiamata dei metodi d'istanza

- Chiamata ai metodi di istanza di **String**:

```
String sampleLower = new String ('a', 5);
String sampleUpper = sampleLower.ToUpper ();

Console.WriteLine (sampleLower); // aaaaa
Console.WriteLine (sampleUpper); // AAAAA
```

- Chiamata ai metodi di istanza di **DateTime**:

```
DateTime now = DateTime.Now;
DateTime later = now.AddHours(8);

Console.WriteLine("Now: {0}", now);
Console.WriteLine("8 hours later: {0}", later);
```

Metodi Statici

I **metodi statici** sono comuni a tutte le istanze di una classe (condivisi tra tutte le istanze)

- Il valore restituito dipende solo dai parametri passati
- Non è disponibile alcuna istanza di classe particolare

Sintassi:

- Il nome della classe, seguito dal nome del metodo, separato da un punto

```
<class_name>.<method_name>(<parameters>)
```

Chiamata Metodi Statici - Esempio

```
using System;

double  raggio = 2.9;
double  area = Math.PI * Math.Pow (raggio, 2);
Console.WriteLine ("Area: {0}", area);
// Area: 26,4207942166902

double  precisione = 8.7654321;
double round3 = Math.Round (precisione, 3);
double round1 = Math.Round (precisione, 1);
Console.WriteLine (
     "{0}; {1}; {2}",  precisione, round3, round1);
// 8,7654321; 8.765; 8,8
```

campo costante

Metodo statico

Metodo statico

Metodo statico

Costruttori

I **costruttori** sono metodi speciali utilizzati per assegnare i valori iniziali dei campi in un oggetto

- Eseguito quando viene creato un oggetto di un determinato tipo
- Hanno lo stesso nome della classe che li detiene
- Nessun return

Una classe può avere diversi costruttori con diversi set di parametri

Costruttori

I costruttori si invocano con l'operatore **new**

```
<instance_name> = new <class_name>(<parameters>)
```

Esempi:

```
String s = new String(new char[]{'a','b','c'});
```

```
String s = new String('*', 5);
```

```
DateTime dt = new DateTime(2009, 12, 30);
```

```
DateTime dt = new DateTime(2009, 12, 30, 12, 33, 59);
```

```
Int32 value = new Int32();
```

Costruttori senza parametri

Il **costruttore senza parametri** è chiamato costruttore predefinito

Esempio:

Creazione di un oggetto per la generazione di numeri casuali

```
using System;
...
Random randomGenerator = new Random();
```

Costruttore senza parametri

System.Randon genera numeri pseudo-casuali

Costruttori con parametri

Esempio:

Creazione di oggetti per la generazione di valori casuali con seed iniziali specificati

```
using System;
...
Random randomGenerator1 = new Random(123);
Console.WriteLine(randomGenerator1.Next());
// 2114319875

Random randomGenerator2 = new Random(456);
Console.WriteLine(randomGenerator2.Next(50));
// 47
```

Costruttori - Esempio

Creazione di un oggetto **DateTime** per una data e un'ora specificate

```
using System;

DateTime halloween = new DateTime(2009, 10, 31);
Console.WriteLine(halloween);

DateTime julyMorning =
  new DateTime(2009,7,1, 5,52,0);
Console.WriteLine(julyMorning);
```

Tra i costruttori definiti, viene chiamato il costruttore con i rispettivi argomenti

Enumerazioni

Le enumerazioni in C # sono tipi i cui valori sono limitati ad un set predefinito di valori.

Esempio: giorni della settimana

Dichiarato dalla parola chiave **enum**

Conserva i valori da un set predefinito

```
public enum Color { Red, Green, Blue, Black }
…
Color color = Color.Red;
Console.WriteLine(color); // Red
color = 5; // Compilation error!
```

Tipo Struct

Le **struct** in C # sono simili alle classi

Le struct sono tipi valore (contengono direttamente un valore)

Le classi sono tipi riferimento (contengono l'indirizzo)

- Le struct vengono solitamente utilizzate per memorizzare strutture dati senza altre funzionalità.

- Le struct possono avere campi, proprietà, ecc.

  L'uso di metodi non è raccomandato

Esempio:

System.DateTime: rappresenta una data e un'ora

I Namespace

I **namespace** vengono utilizzati per organizzare il codice sorgente in modo più logico e gestibile.

I namespace possono contenere:

Definizioni di classi, structs, interfacce, altri tipi e altri namespace

I namespace possono contenere altri namespace.

Per esempio:

Lo spazio dei nomi di sistema contiene il namespace dei dati

Il nome di un namespace nidificato è **System.Data**

Nome completo della Classe

Un nome completo di una classe è il nome della classe preceduto dal nome del suo namespace

```
<namespace_name>.<class_name>
```

Esempio:

Classe **Array**, definita nello spazio dei nomi **System**

Il nome completo della classe è **System.Array**

La direttiva using

La direttiva **using** in C #: `using <namespace_name>`

Consente l'utilizzo di tipi in un namespace, senza specificarne il nome completo.

Esempio:

```
using System;
DateTime date;
```

invece di

```
System.DateTime date;
```

La classe Random

La classe **Random** genera numeri interi casuali.

```
Random rand = new Random();
for (int number = 1; number <= 6; number++)
{
   int randomNumber = rand.Next(49) + 1;
   Console.Write("{0} ", randomNumber);
}
```

L'esempio genera 6 **int casuali** nell'intervallo [1..49]

Usare sempre una singola istanza Random, ciò eviterà anomalie

Common Type System

CTS definisce tutti i tipi di dati supportati in .NET Framework

Tipi primitivi (ad esempio int, float, object)
Classi (ad es. String, Console, Array)
Structs (ad esempio DateTime)
Array (ad es. Int [], string [,])
...

CTS e Differenti Linguaggi

CTS è comune a tutti i linguaggi .NET

C #, VB.NET, J #, JScript.NET, ...

- Mappature del tipo nel **CTS**:

CTS Type	C# Type	VB.NET Type
System.Int32	int	Integer
System.Single	float	Single
System.Boolean	bool	Boolean
System.String	string	String
System.Object	object	Object

System.Object: CTS base Type

System.Object è il tipo base per tutti gli altri tipi nel CTS

Può contenere valori di qualsiasi altro tipo:

```
string s = "test";
object obj = s;
```

Tutti i tipi .NET derivano metodi comuni da **System.Object**, ad es. ToString()

```
DateTime now = DateTime.Now;
string nowInWords = now.ToString();
Console.WriteLine(nowInWords);
```

Tipi valore e tipi riferimento

Nel **CTS** sono presenti due categorie di tipi...

Tipi valore

Tipi riferimento

...collocate in diverse aree della memoria:

I tipi valore risiedono nell'**execution stack**

Liberato quando diventa fuori scope

Tipi valore e tipi riferimento

I tipi di riferimento risiedono nel **managed heap**(memoria dinamica)

Liberato dal garbage collector

- * **Nota**: questo non significa che i tipi valore, che fanno parte dei tipi riferimento, risiedono nello stack. Ad esempio, gli interi in un **List<int>** non risiedono nello stack.

Tipi valore e tipi riferimento

Tipi valore:

- La maggior parte dei tipi primitivi
- Struct
- Esempi: **int**, **float**, **bool**, **DateTime**

Tipi di riferimento:

- Classi e interfacce
- stringhe
- Array
- Esempi: **string**, **Random**, **object**, **int[]**

Tipi parziali

I **tipi parziali** consentono di suddividere il codice di implementazione di un tipo in più file.

Può essere utile quando il numero di istruzioni presenti è talmente alto da essere più facilmente gestibile in diversi file, separando eventualmente in diverse sezioni per aree di funzionalità differenti.

Tipi parziali

Per creare tipi parziali basta utilizzare il modificatore **partial** prima della definizione della classe.

Ogni parte del file deve comprendere il modificatore partial e i modificatori di accesso non devono essere in conflitto

```
//File class1.cs
partial class Class1
{
//membri
}
//secondo_File.cs
partial class Class1
{ //altri membri
}
```

Tipi parziali

Ogni tipo parziale può specificare una classe base o interfacce indipendenti da implementare per ognuna delle parti:

```
//File class1.cs
partial class Class1: ClasseBase
{
//membri
}
//secondo_File.cs
partial class Class1: Interfaccia1
{ //altri membri
}
```

Tipi parziali

Un tipo parziale può contenere metodi parziali: metodi la cui implementazione è suddivisa in una parte di definizione, che indica solo la firma del metodo, e una parte con l'implementazione vera e propria del corpo del metodo.

```
//File class1.cs
partial class Class1
{
partial void MetodoParziale();
}
//secondo_File.cs
partial class Class1
{
partial void MetodoParziale()
{
//implementazione del metodo
}
}
```

Tipi anonimi

La parola chiave **var** permette di definire **variabili di tipo implicito**.

Utilizzandola in congiunzione con la parola chiave new è possibile creare delle **variabili di tipo anonimo**.

I tipi anonimi forniscono un modo per definire una classe dichiarandone le proprietà ma senza definirla esplicitamente.

Tipi anonimi

Un tipo anonimo è una **classe senza nome**, derivata da **System.Object**, la cui struttura è ricavata implicitamente dal compilatore :

```
var cliente = new {Nome="mario", Cognome="rossi", Età=50};
```

La parola chiave **new** è seguita da un blocco chiamato **inizializzatore dell'oggetto**, nel quale vengono definite e assegnate le proprietà.

Il compilatore genererà una classe con due proprietà Nome e Cognome di tipo String e una proprietà Età di tipo int.

Tipi anonimi

Successivamente la variabile cliente è utilizzabile normalmente, per esempio è possibile stamparne i valori in console:

```
string nome = cliente.Nome;
Console.WriteLine("{0} {1}, anni {2}", cliente.Nome, cliente.Cognome, cliente.Età);
```

Il tipo anonimo è utile per creare un contenitore da utilizzate temporaneamente in un blocco di codice, per esempio all'interno di un metodo:; infatti, non avendo un tipo dichiarato, non è possibile utilizzarli come parametri di altri metodi o come valori di ritorno.

Domande e approfondimenti

Academy .NET