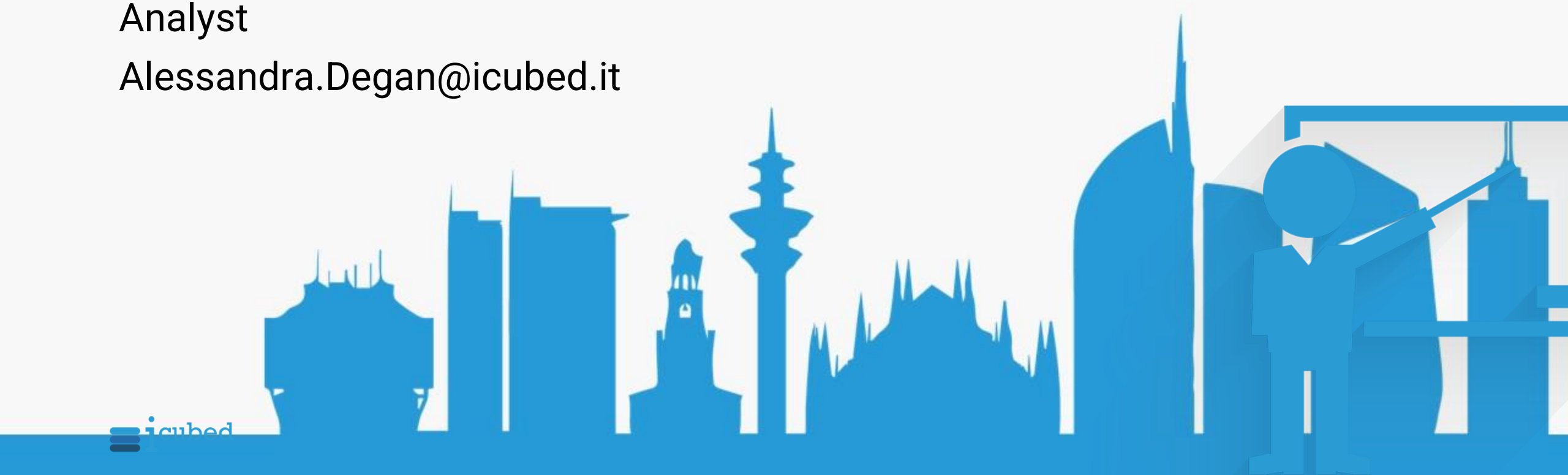
Programmazione a oggetti

Alessandra Degan Di Dieco

Analyst

Alessandra.Degan@icubed.it



Proprietà di altro tipo definito dall'utente

Cos'è la proprietà Teacher?

```
public class Student
{
    public int Id { get; set; }

    public string FirstName { get; set; }
    public string LastName { get; set; }

    public Teacher Teacher { get; set; }
}
```

Può essere anche una Navigation Property.

"Primary Key"

```
public class Student
{
      public int Id { get; set; }

      public string FirstName { get; set; }
      public string LastName { get; set; }

      public Teacher Teacher { get; set; }
}
```

Chiave primaria → Identificativo univoco

Es. Id (int), Codice (string)

Ricorda!!! Per il momento non abbiamo un vero database sotto!!!

icubed

”Foreign Key”

```
public class Student
{
      public int Id { get; set; }

      public string FirstName { get; set; }
      public string LastName { get; set; }

      public int CourseId { get; set; } //FK
      public Course Course { get; set; }
                              //Navigation Property
}
```

```
public class Course
{
      public int Id { get; set; }

      public string Name { get; set; }

      public Teacher Teacher { get; set; }

      public List<Student> Students { get; set; }
}
```

Un corso può avere più studenti.

Uno studente può essere iscritto a un solo corso.

Ricorda!!! Per il momento non abbiamo un vero database sotto!!!

Relazioni uno-a-molti

Un insegnante può tenere più di un corso

Ricorda!!! **Per il momento non abbiamo un vero database sotto!!!**

```
public class Course
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int TeacherId { get; set; }
    public  Teacher Teacher { get; set; }
}
```

```
public class Teacher
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public List<Course> Courses { get; set; } = new List<Course>();
}
```

Relazioni uno-a-molti

Un insegnante può tenere più di un corso

Ricorda!!! **Per il momento non abbiamo un vero database sotto!!!**

```
public class Course
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int TeacherId { get; set; }
    public  Teacher Teacher { get; set; }
}
```

```
public class Teacher
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public List<Course> Courses { get; set; } = new List<Course>();
}
```

Relazioni uno-a-molti

Un insegnante può tenere più di un corso

Ricorda!!! **Per il momento non abbiamo un vero database sotto!!!**

```
public class Course
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int TeacherId { get; set; }
    public Teacher Teacher { get; set; }
}
```

```
public class Teacher
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public List<Course> Courses { get; set; } = new List<Course>();
}
```

Relazioni uno-a-uno

Una relazione uno-a-uno è, per definizione, una relazione per cui la chiave primaria di una tabella diventa chiave primaria e chiave esterna dell'altra

```
public class Student
{
    public int Id { get; set; }

     public string FirstName { get; set; }
     public string LastName { get; set; }

    public Address Address { get; set; } }
```

```
public class Address
{
    public string Street { get; set; }
    public string City { get; set; }

    public int StudentId { get; set; }
    public Student Student { get; set; }
}
```

Relazioni molti-a-molti

Uno studente è iscritto a più corsi e ogni corso può avere più studenti

```
public class Course
{
    public int Id { get; set; }
    public string Name { get; set; }

    public List<Student> Students { get; set; }
}
```

```
public class Student
{
     public int Id { get; set; }

     public string FirstName { get; set; }
     public string LastName { get; set; }

     public List<Course> Courses { get; set; }

}
```

Ricorda!!! Per il momento non abbiamo un vero database sotto!!!

icubed

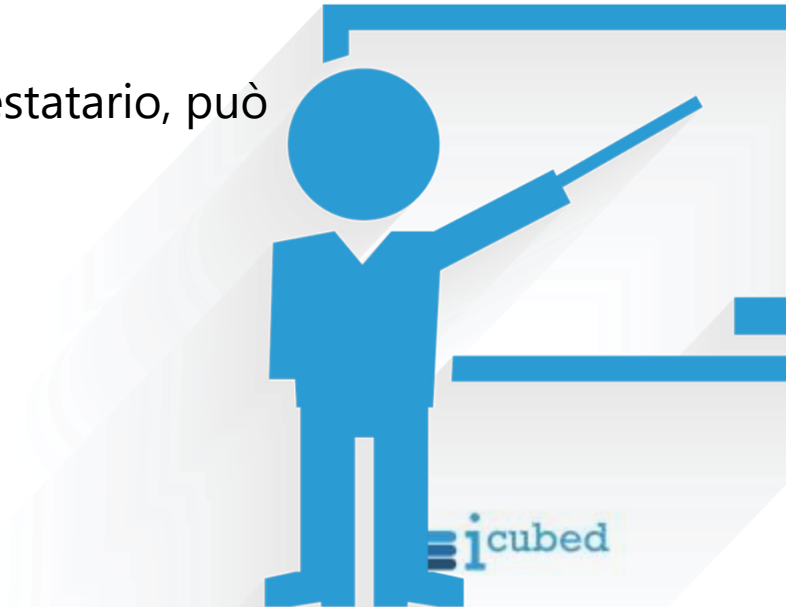
Esercitazione 3

Scrivere un programma per la gestione dei conti in una banca.

Un conto è un'entità che ha
- Numero di conto (int)
- Saldo (decimal)

Un intestatario è un’entità che ha
- Codice cliente
- Nome
- Cognome

Il conto può avere UN SOLO intestatario. Il cliente della banca, ovvero l’intestatario, può avere più conti nella banca.



Esercitazione 3

L'utente banchiere può:
1) Creare un nuovo conto
2) Gestire il prelievo da un conto
3) Gestire il versamento su un conto
4) Visualizzare il saldo di un conto
5) Eliminare un conto

1) Creare un nuovo conto
Se il cliente è nuovo, bisogna aggiungerlo tra i clienti della banca e poi aggiungere il nuovo conto sul cliente.

Se il cliente è un vecchio cliente, bisogna aggiungere un conto su un cliente già esistente.

Quando viene creato un nuovo conto, il saldo iniziale è 0.

Esercitazione 3

Il numero di conto deve essere autogenerato, ovvero non deve essere scelto dall'utente banchiere, deve essere diverso da quelli già esistenti.

Esempi di soluzione:

- Assegnare l'indice dell'ultimo conto nell'elenco + 1
- Generare un numero di conto random (ma, all'inserimento, assicurarsi che non ci sia già nell'elenco un conto con quel numero di conto)

Nullable

- Le variabili di tipo Type Reference **possono essere** null
- Le variabili di tipo Value **non possono essere** null

- Problema tipico: mappatura tra classi e tabelle del database
  - I valori nel db possono essere **null**
  - I valori dei campi delle classi non lo possono essere (es. Int, double)

Nullable

- In C# possiamo utilizzare i **tipi Nullable**

➔ tipi che **possono essere null**

- E’ sufficiente utilizzare il **carattere “?”** dopo un tipo per renderlo nullabile:

```
int x1 = 1; // intero "tradizionale"
int? x2 = null; // intero "nullabile"
```

- L’assegnazione x1 a x2 non genera problemi
- L’assegnazione da x2 a x1 non è possibile. E’ necessario il cast:

```
x1 = (int) x2
```

Nullable

- Il tipo nullabile ha anche associato il **metodo .HasValue()** che ritorna true o false a seconda che il tipo abbiamo un valore o meno.
- Nella **proprietà .Value** possiamo recuperare il valore

Demo

Nullable

Convenzioni sul codice

- **Notazione ungherese:** al nome dell'identificatore viene aggiunto un prefisso che ne indica il tipo (es. intNumber identifica una variabile intera)

- **Notazione Pascal:** l'inizio di ogni parola che compone il nome dell'identificatore è maiuscola, mentre tutte le altre lettere sono minuscule (es. FullName)

- **Notazione Camel:** come la notazione Pascal, a differenza del fatto che la prima iniziale deve essere minuscola (es. fullName)

Convenzioni sul codice

Elementi	Notazione
Namespace	Notazione Pascal
Classi	Notazione Pascal
Interfacce	Notazione Pascal
Strutture	Notazione Pascal
Enumerazioni	Notazione Pascal
Campi privati	Notazione Camel
Proprietà, metodi e eventi	Notazione Pascal
Parametri di metodi e funzioni	Notazione Camel
Variabili locali	Notazione Camel

© 2019 iCubed Srl

iCubed s.r.l. • Piazza Durante, 8 – 20131, Milano
• Phone: +39 02 57501057 • P.IVA 07284390965