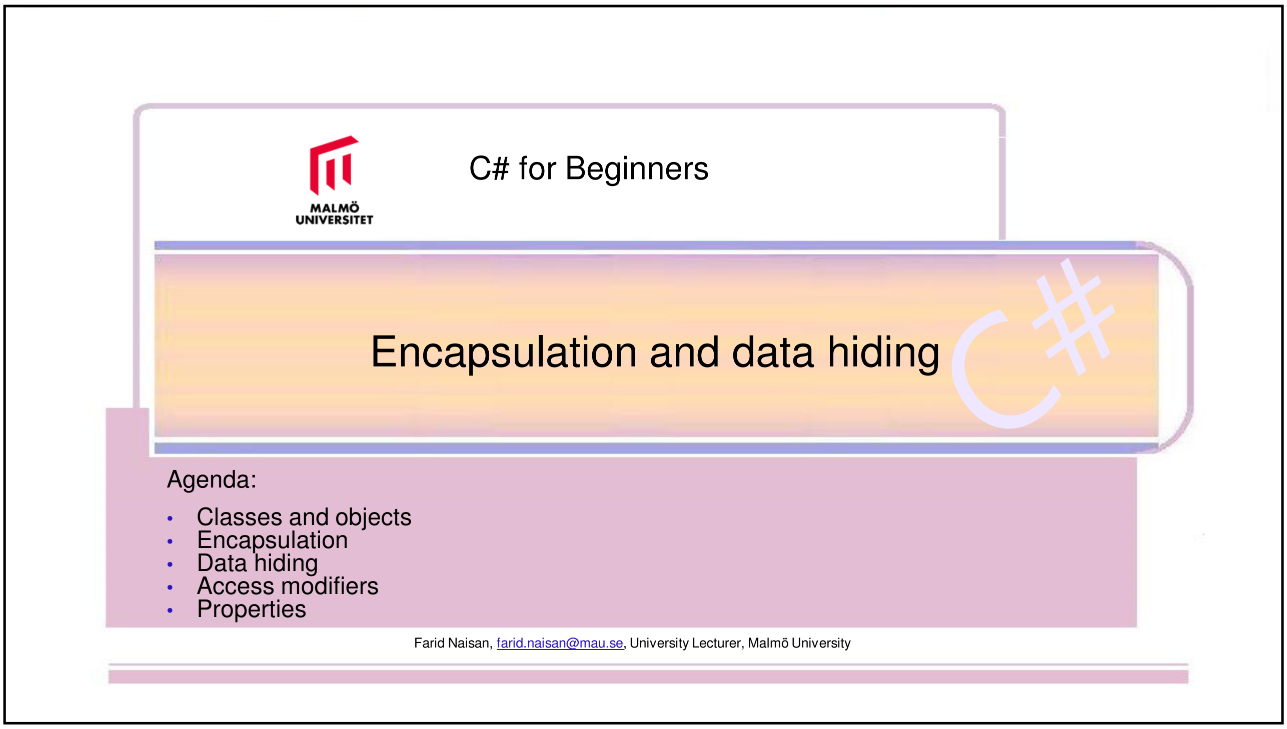


MALMÖ UNIVERSITET

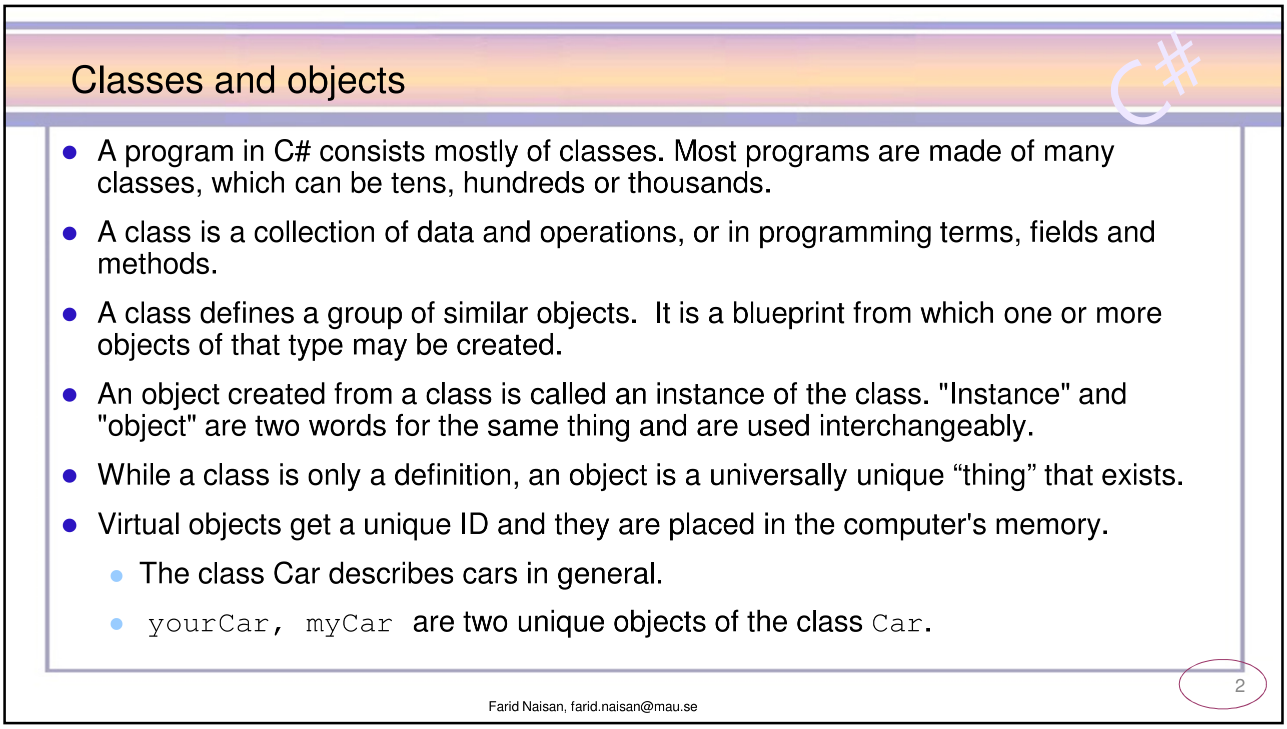
C# for Beginners

Encapsulation and data hiding

Agenda:

- Classes and objects
- Encapsulation
- Data hiding
- Access modifiers
- Properties

Farid Naisan, farid.naisan@mau.se, University Lecturer, Malmö University



Classes and objects

- A program in C# consists mostly of classes. Most programs are made of many classes, which can be tens, hundreds or thousands.
- A class is a collection of data and operations, or in programming terms, fields and methods.
- A class defines a group of similar objects. It is a blueprint from which one or more objects of that type may be created.
- An object created from a class is called an instance of the class. "Instance" and "object" are two words for the same thing and are used interchangeably.
- While a class is only a definition, an object is a universally unique “thing” that exists.
- Virtual objects get a unique ID and they are placed in the computer's memory.
  - The class Car describes cars in general.
  - `yourCar, myCar` are two unique objects of the class `Car`.

2

Farid Naisan, farid.naisan@mau.se

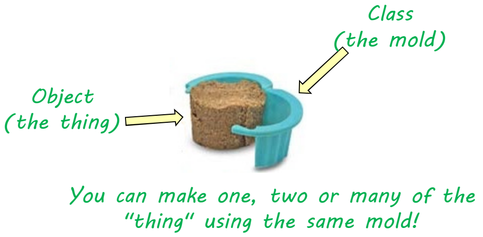
Class and object

- Classes can represent physical objects like `Car, House, TV`, or conceptual objects like `Address, BankLoan,` or `Rectangle`.
- A class is a type while an object in the memory that is created from the class and is referenced by a reference variable.
- Objects are created by the keyword new and placed on the heap memory.
- A reference variable contains the address of an object.

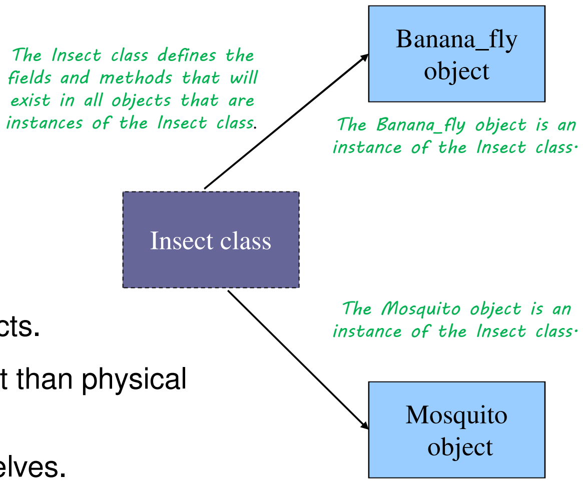
```
    BankLoan loan = new BankLoan();
```

- It is very common to use the word "object" to a reference variable, which actually only has the address of the memory where the object is placed.
- Every instance of a class will have its own set of the values stored in the fields of the object.
- Every instance of a class will have a copy of the methods of the class.

Classes and Objects



You can make one, two or many of the "thing" using the same mold!



The Insect class defines the fields and methods that will exist in all objects that are instances of the Insect class.

The Banana_fly object is an instance of the Insect class.

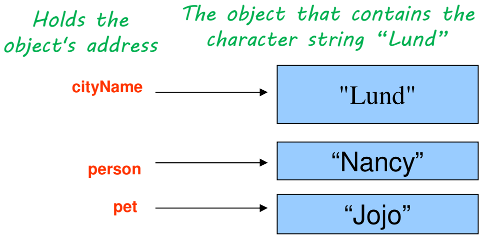
The Mosquito object is an instance of the Insect class.

- Class is like a mold used to create objects.
- Virtual objects are much more intelligent than physical objects.
- They can perform operations on themselves.

Classes

- Many objects can be created from a class. Each object can be accessed (referred to) using a reference variable.
- A reference variable contains the address of an object in the memory.

```
string cityName = "Lund";
string person ="Nancy";
string pet = "Jojo";
```

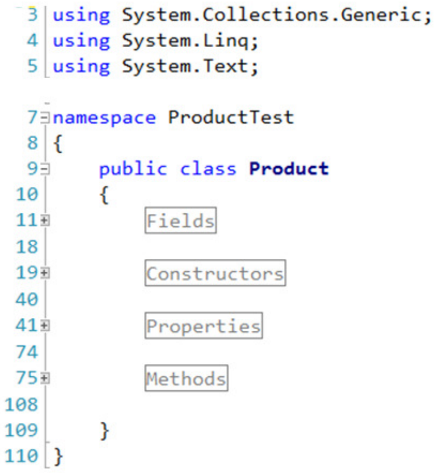

Holds the object's address
The object that contains the character string "Lund"
cityName
"Lund"
person
"Nancy"
pet
"Jojo"

- Class objects normally have properties and methods that perform useful operations on their data.
- The Length property of the string class returns an integer value that is equal to the length of the string.

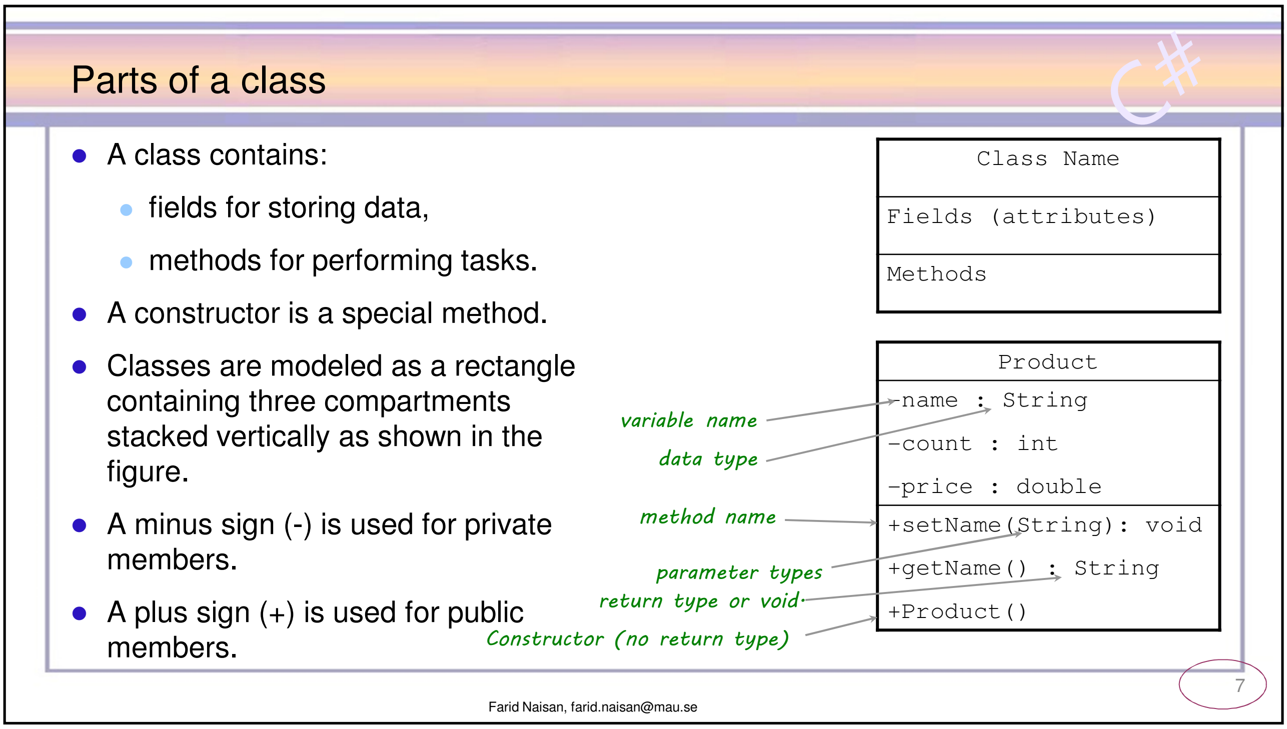
```
int stringLength = cityName.Length;
```

Class definition

- A class is defined using the keyword class followed by a class name that must abide with the identifier naming rules.
- A general layout of a class may look as in the figure.
- At the top of the file, namespaces containing classes we need to use are included.
- A namespace is like a package name for the collection of types included in the in the package.
- Namespaces allow you to write classes with the same name but in different namespaces.
- Whenever the compiler does not recognize a class, it can happen that the namespace under which the class is located is not imported.
- It is always a good idea to use a namespace for every application.



```
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ProductTest
{
    public class Product
    {
        Fields

        Constructors

        Properties

        Methods

    }
}
```

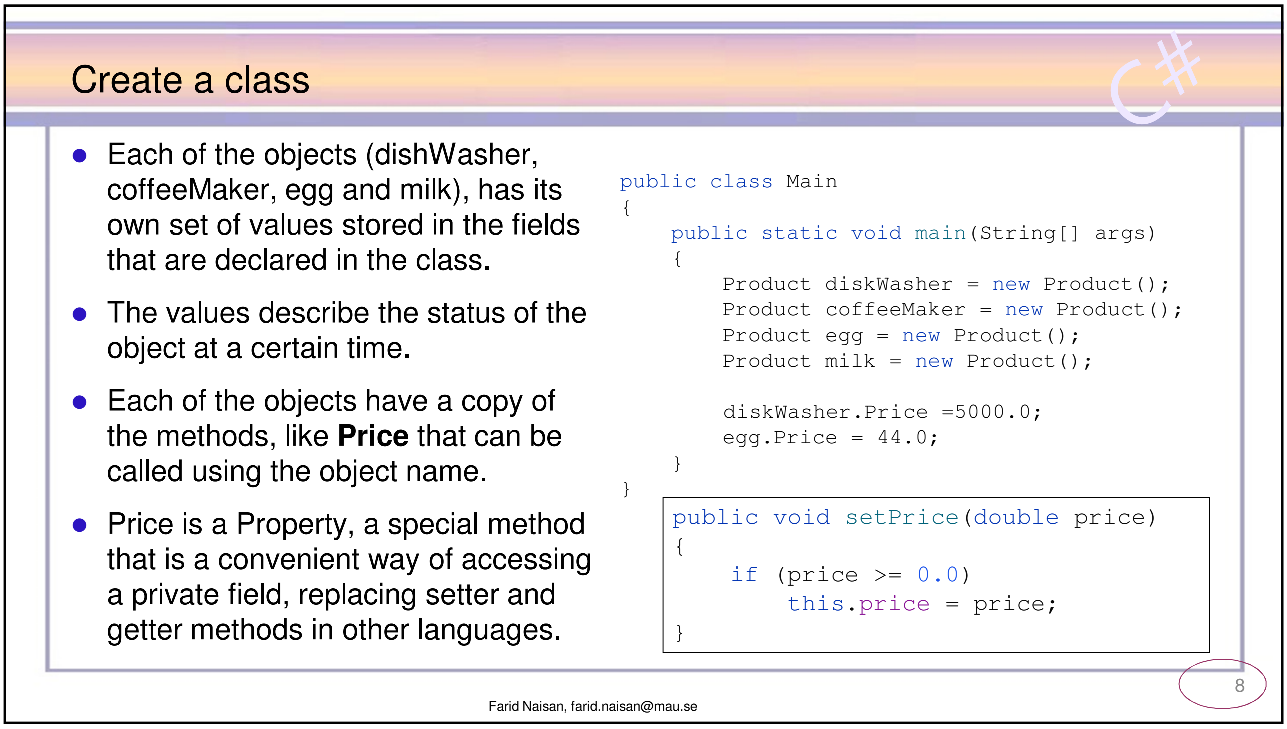


Parts of a class

- A class contains:
  - fields for storing data,
  - methods for performing tasks.
- A constructor is a special method.
- Classes are modeled as a rectangle containing three compartments stacked vertically as shown in the figure.
- A minus sign (-) is used for private members.
- A plus sign (+) is used for public members.

```
Class Name
Fields (attributes)
Methods
```

```
Product
-name : String
-count : int
-price : double
+setName(String): void
+getName() : String
+Product()
```

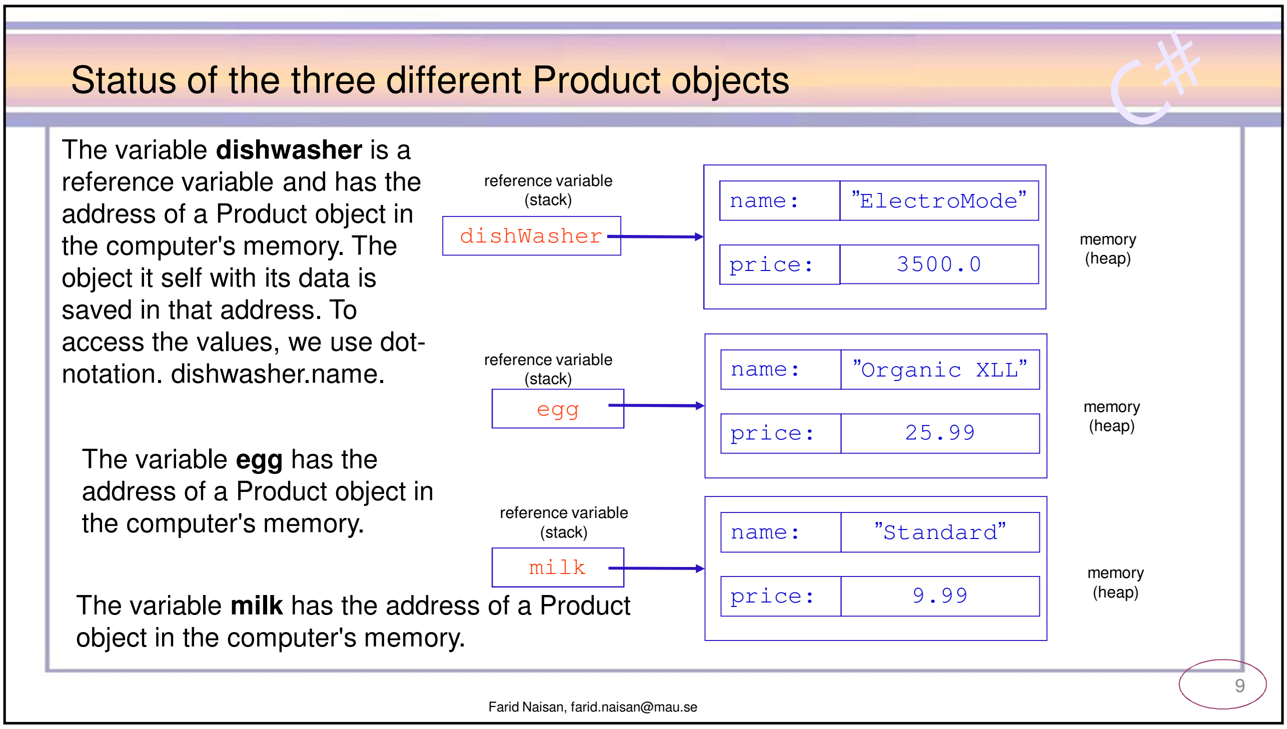


Create a class

- Each of the objects (dishWasher, coffeeMaker, egg and milk), has its own set of values stored in the fields that are declared in the class.
- The values describe the status of the object at a certain time.
- Each of the objects have a copy of the methods, like **Price** that can be called using the object name.
- Price is a Property, a special method that is a convenient way of accessing a private field, replacing setter and getter methods in other languages.

```
public class Main
{
    public static void main(String[] args)
    {
        Product diskWasher = new Product();
        Product coffeeMaker = new Product();
        Product egg = new Product();
        Product milk = new Product();

        diskWasher.Price =5000.0;
        egg.Price = 44.0;
    }
}
```

```
public void setPrice(double price)
{
    if (price >= 0.0)
        this.price = price;
}
```

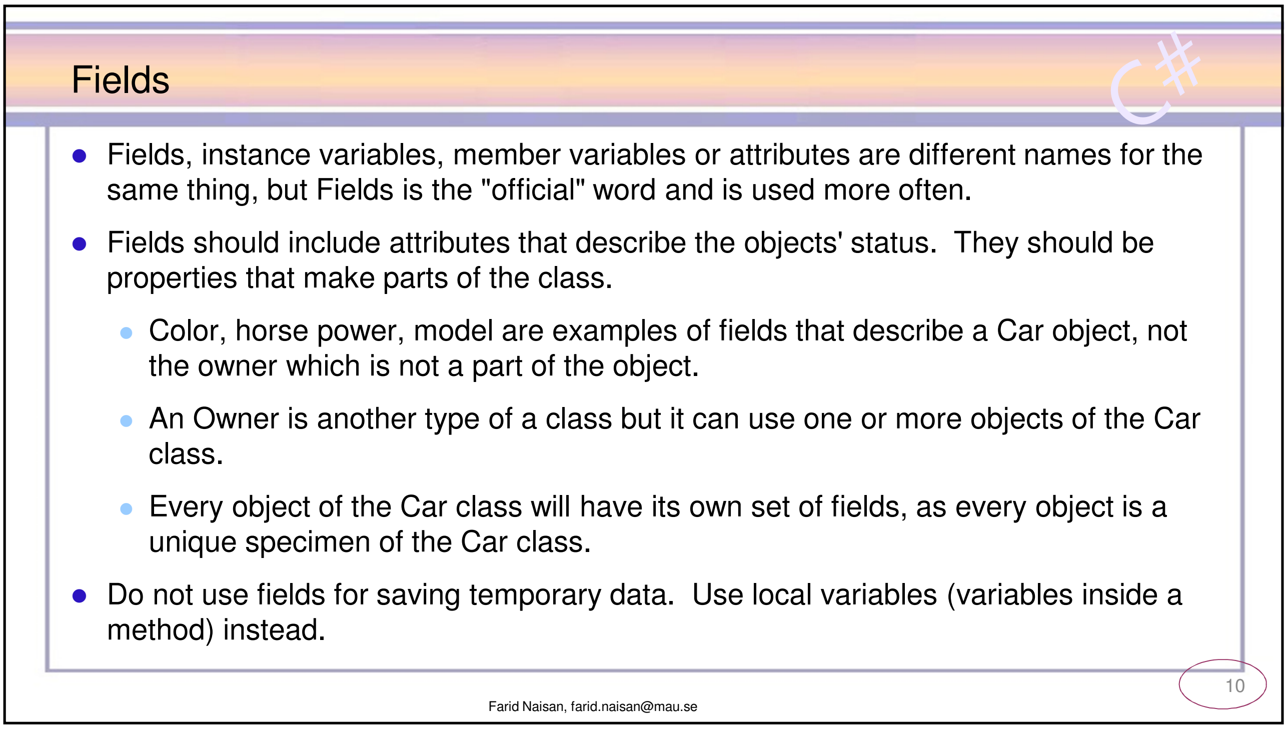


Status of the three different Product objects

The variable **dishwasher** is a reference variable and has the address of a Product object in the computer's memory. The object it self with its data is saved in that address. To access the values, we use dot-notation. dishwasher.name.

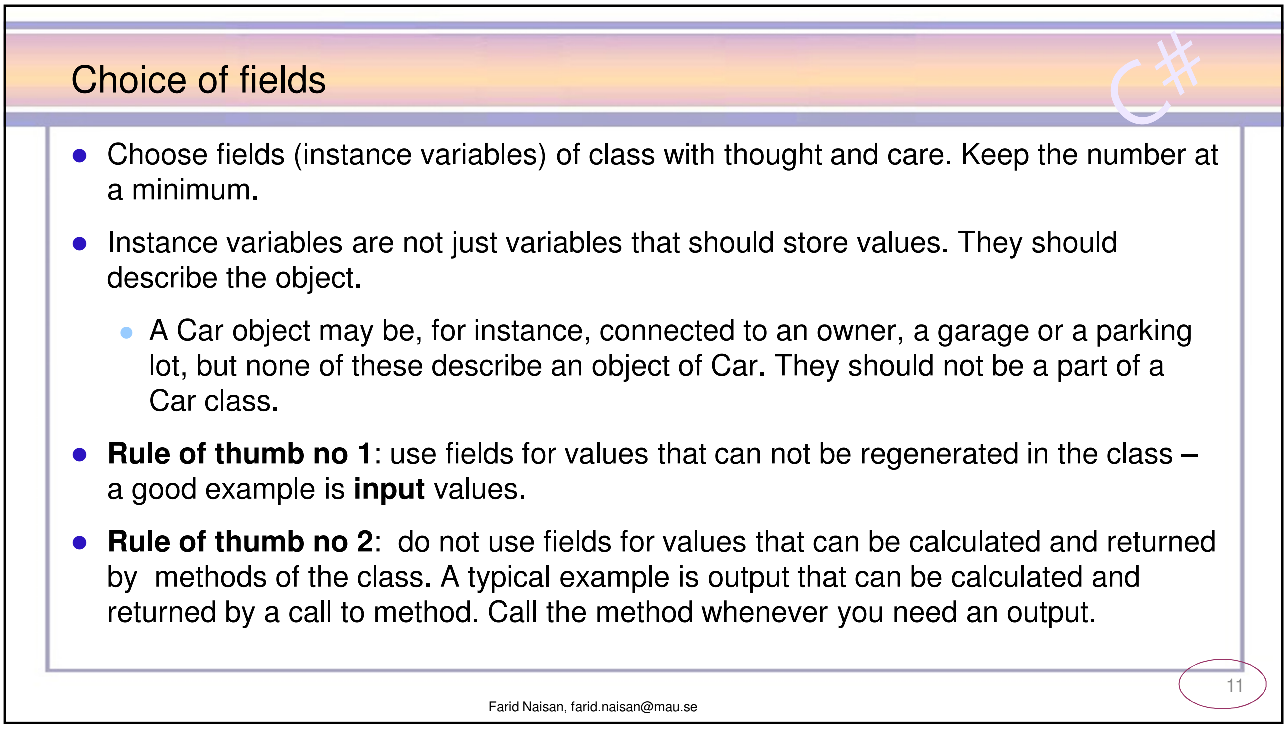
The variable **egg** has the address of a Product object in the computer's memory.

The variable **milk** has the address of a Product object in the computer's memory.



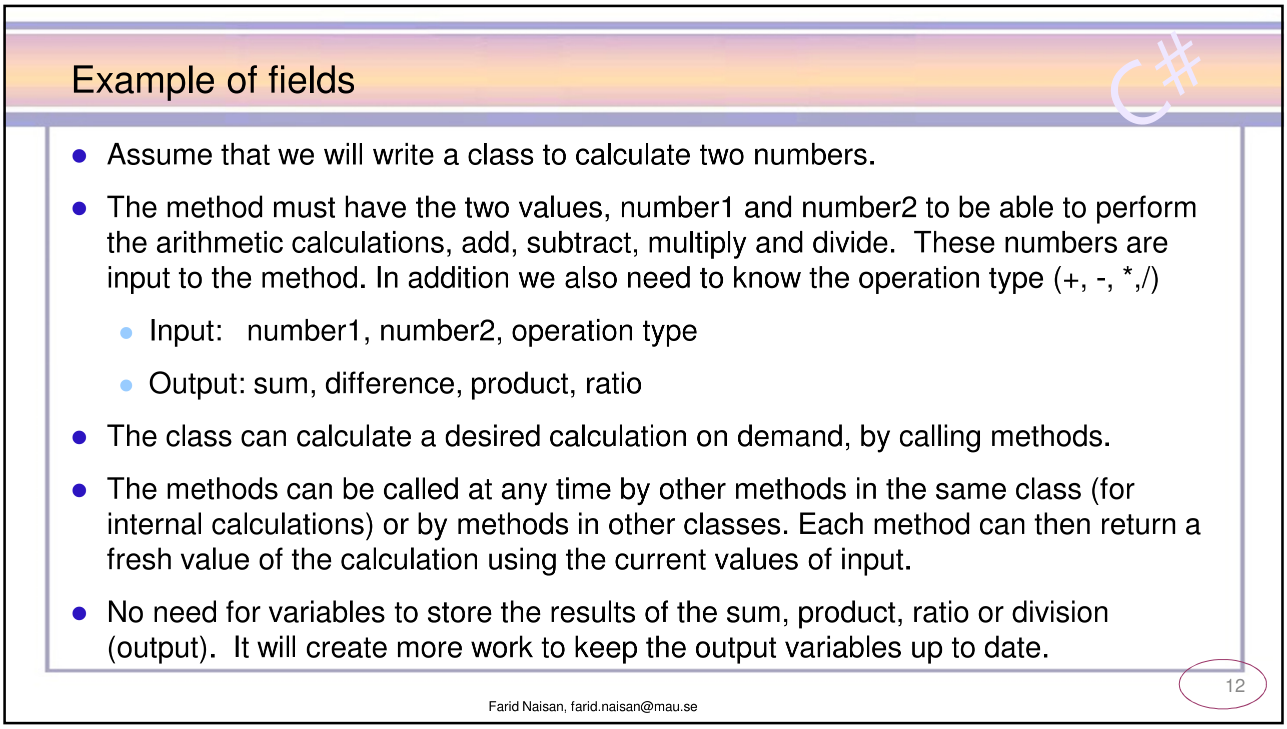
Fields

- Fields, instance variables, member variables or attributes are different names for the same thing, but Fields is the "official" word and is used more often.
- Fields should include attributes that describe the objects' status. They should be properties that make parts of the class.
  - Color, horse power, model are examples of fields that describe a Car object, not the owner which is not a part of the object.
  - An Owner is another type of a class but it can use one or more objects of the Car class.
  - Every object of the Car class will have its own set of fields, as every object is a unique specimen of the Car class.
- Do not use fields for saving temporary data. Use local variables (variables inside a method) instead.



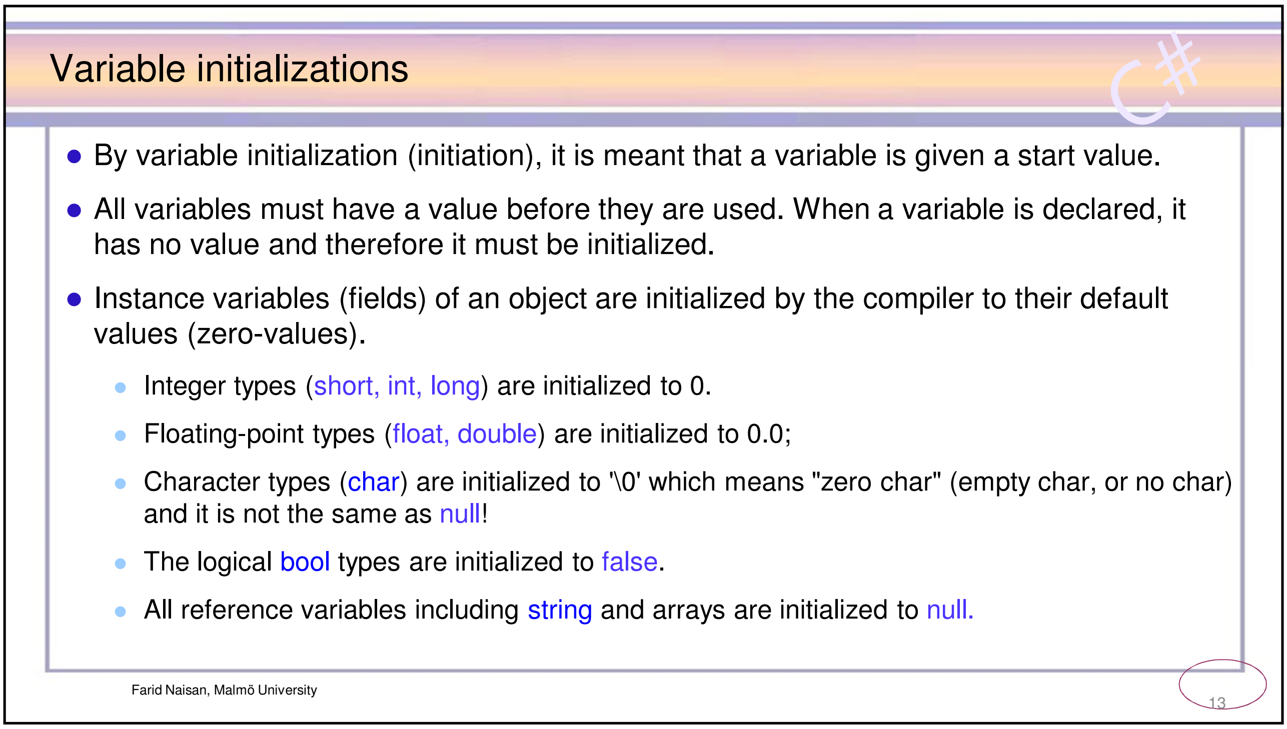
Choice of fields

- Choose fields (instance variables) of class with thought and care. Keep the number at a minimum.
- Instance variables are not just variables that should store values. They should describe the object.
  - A Car object may be, for instance, connected to an owner, a garage or a parking lot, but none of these describe an object of Car. They should not be a part of a Car class.
- **Rule of thumb no 1**: use fields for values that can not be regenerated in the class – a good example is **input** values.
- **Rule of thumb no 2**: do not use fields for values that can be calculated and returned by methods of the class. A typical example is output that can be calculated and returned by a call to method. Call the method whenever you need an output.



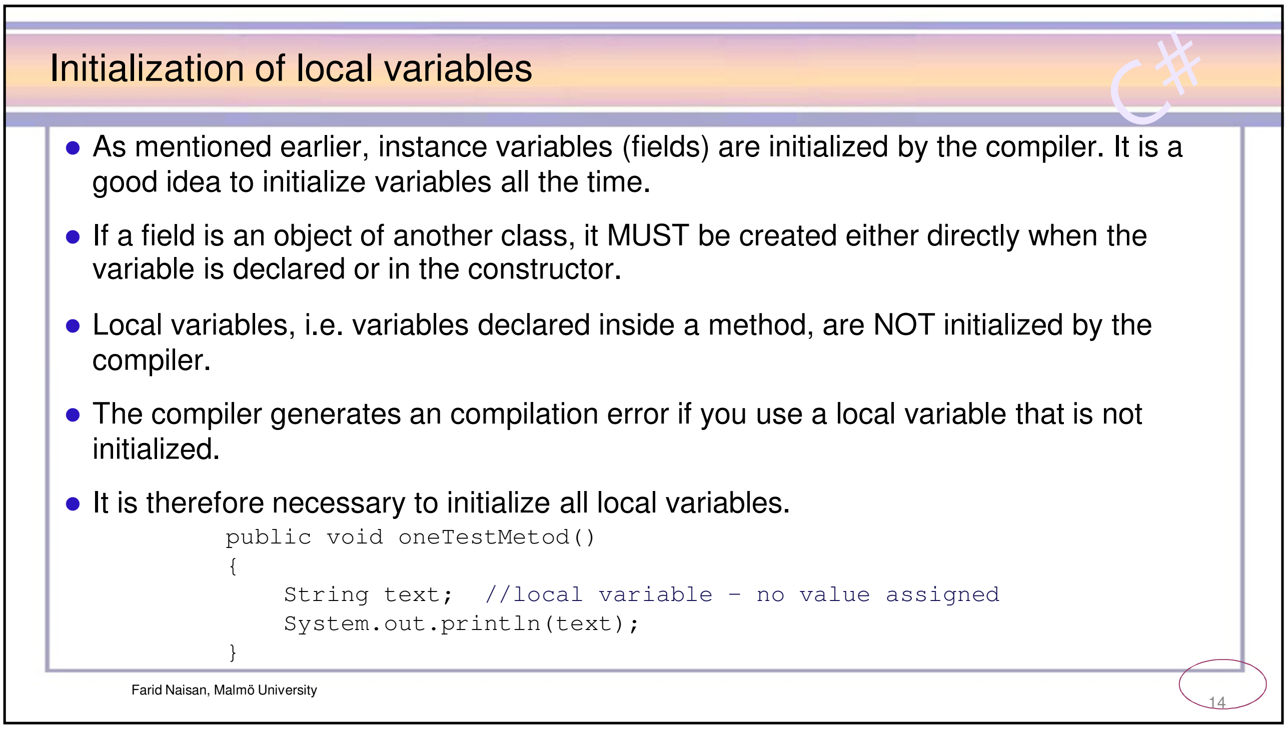
Example of fields

- Assume that we will write a class to calculate two numbers.
- The method must have the two values, number1 and number2 to be able to perform the arithmetic calculations, add, subtract, multiply and divide. These numbers are input to the method. In addition we also need to know the operation type (+, -, *,/)
  - Input: number1, number2, operation type
  - Output: sum, difference, product, ratio
- The class can calculate a desired calculation on demand, by calling methods.
- The methods can be called at any time by other methods in the same class (for internal calculations) or by methods in other classes. Each method can then return a fresh value of the calculation using the current values of input.
- No need for variables to store the results of the sum, product, ratio or division (output). It will create more work to keep the output variables up to date.



Variable initializations

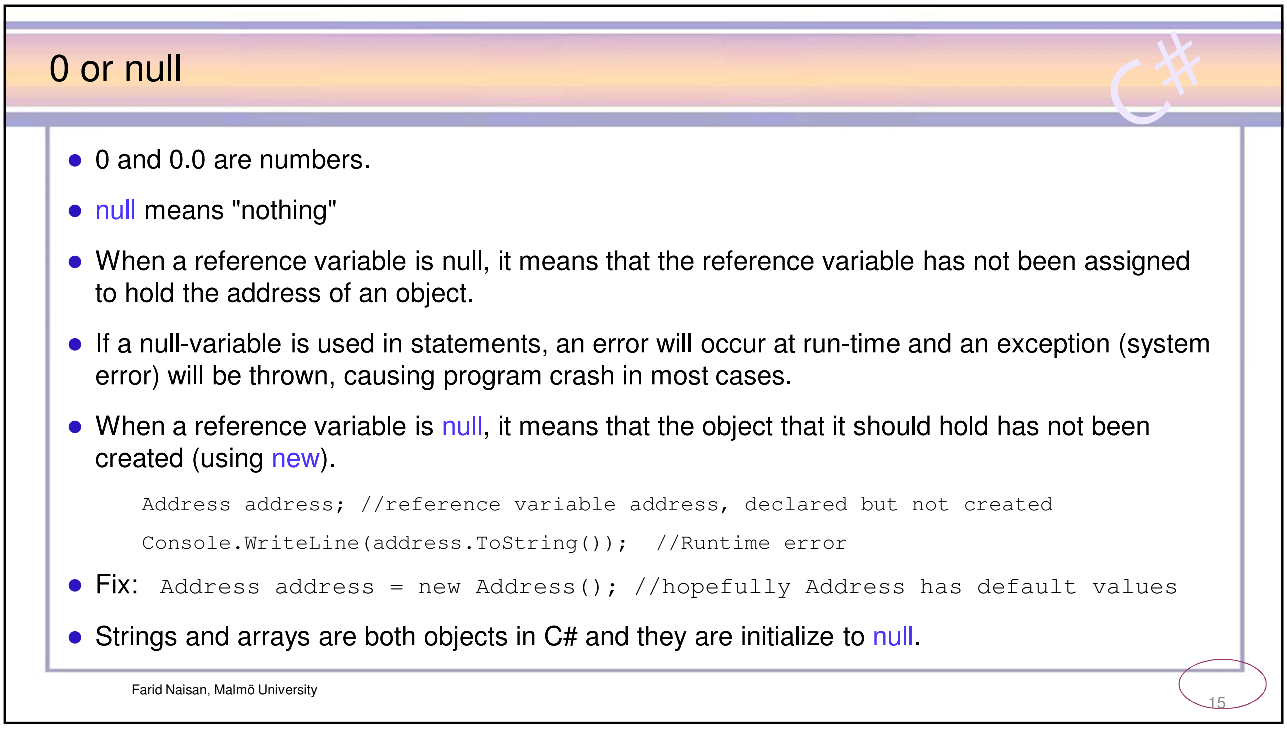
- By variable initialization (initiation), it is meant that a variable is given a start value.
- All variables must have a value before they are used. When a variable is declared, it has no value and therefore it must be initialized.
- Instance variables (fields) of an object are initialized by the compiler to their default values (zero-values).
  - Integer types (short, int, long) are initialized to 0.
  - Floating-point types (float, double) are initialized to 0.0;
  - Character types (char) are initialized to '\0' which means "zero char" (empty char, or no char) and it is not the same as null!
  - The logical bool types are initialized to false.
  - All reference variables including string and arrays are initialized to null.



Initialization of local variables

- As mentioned earlier, instance variables (fields) are initialized by the compiler. It is a good idea to initialize variables all the time.
- If a field is an object of another class, it MUST be created either directly when the variable is declared or in the constructor.
- Local variables, i.e. variables declared inside a method, are NOT initialized by the compiler.
- The compiler generates an compilation error if you use a local variable that is not initialized.
- It is therefore necessary to initialize all local variables.

```
public void oneTestMetod()
{
    String text;  //local variable - no value assigned
    System.out.println(text);
}
```

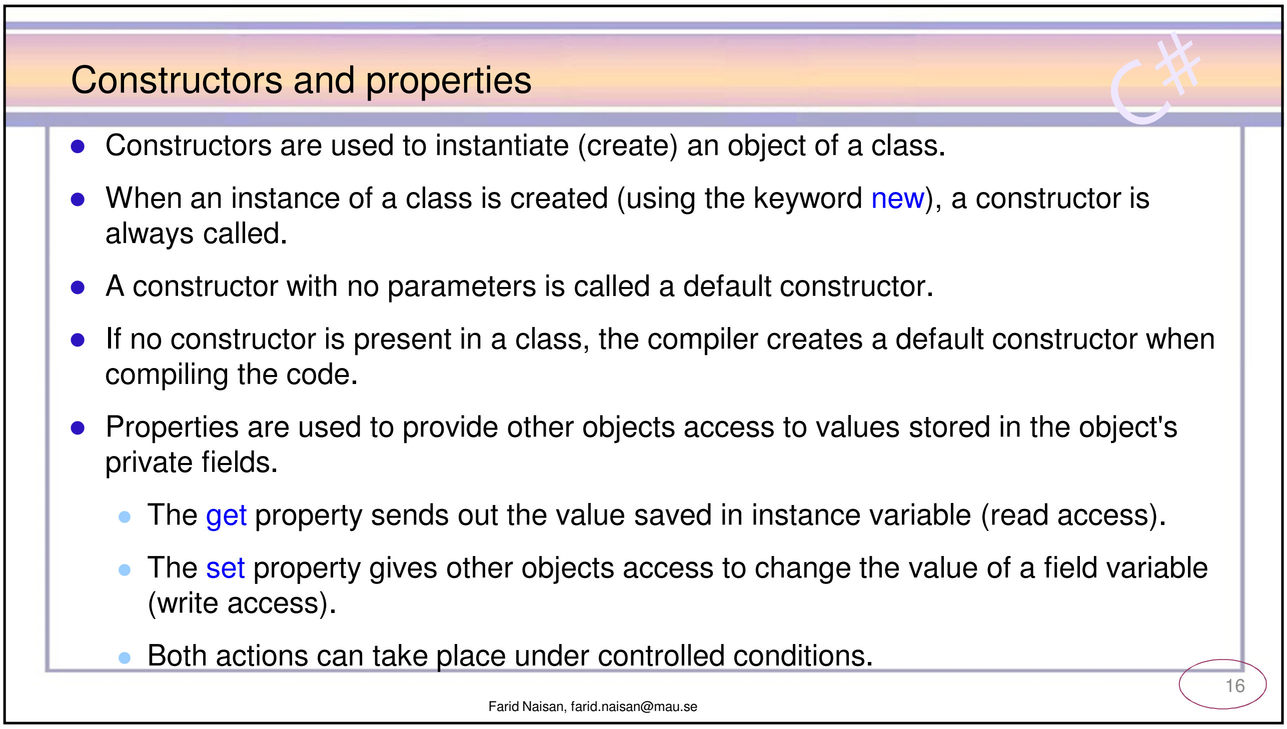


0 or null

- 0 and 0.0 are numbers.
- null means "nothing"
- When a reference variable is null, it means that the reference variable has not been assigned to hold the address of an object.
- If a null-variable is used in statements, an error will occur at run-time and an exception (system error) will be thrown, causing program crash in most cases.
- When a reference variable is null, it means that the object that it should hold has not been created (using new).

```
Address address; //reference variable address, declared but not created
Console.WriteLine(address.ToString());  //Runtime error
```

- Fix: `Address address = new Address(); //hopefully Address has default values`
- Strings and arrays are both objects in C# and they are initialize to null.



Constructors and properties

- Constructors are used to instantiate (create) an object of a class.
- When an instance of a class is created (using the keyword new), a constructor is always called.
- A constructor with no parameters is called a default constructor.
- If no constructor is present in a class, the compiler creates a default constructor when compiling the code.
- Properties are used to provide other objects access to values stored in the object's private fields.
  - The get property sends out the value saved in instance variable (read access).
  - The set property gives other objects access to change the value of a field variable (write access).
  - Both actions can take place under controlled conditions.

Example - Building a `Rectangle` class

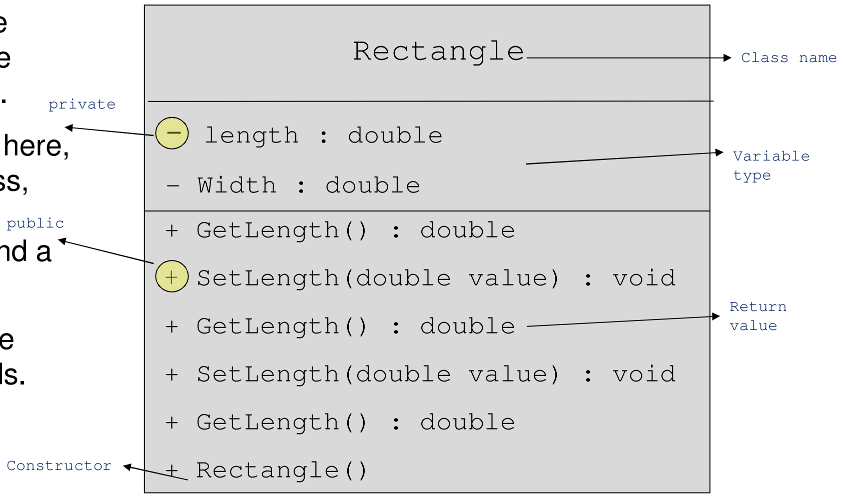
- When constructing a class, the first thing to think about is the purpose of the class. In other words in which context the class is going to be used.
  - This will affect the choice of fields and methods of the class.
- If you have to write a class Car, you should immediately think about who and for which purpose the Car class will be used.
  - An auto-dealer is interested in the model, price, color of a car.
  - An mechanic is interested in the model, details of the engine, not the price.
- As an example, let's write a class for Rectangle for which we would like to calculate the circumference (sum of sides) and area. A rectangle in this case is a geometric shape.
- Determine the input and output:
  - Input: length and width
  - Output: Circumference and area

Rectangle class

- A `Rectangle` object needs to have the following fields:
  - `length`. The length field will store the value of the rectangle's length.
  - `width`. The width field will hold the rectangle's width.
- The `Rectangle` class will also have the following methods:
  - `SetLength`: To store a value in an object's length field.
  - `SetWidth`: to store a value in an object's width field.
  - `GetLength`: this method will return the value in an object's length field.
  - `GetWidth`: This method will return the value in an object's width field.
  - `GetArea`: This method will return the area of the rectangle, which is the result of the object's length multiplied by its width.

UML Diagram for `Rectangle` class

- A class model can be drawn to visualize the structure of the class.
- Based on the sketch here, you can create a class, with private fields, a default constructor and a number of methods.
- Use Properties for the Get- and Set-methods.



Rectangle

− length : double

− Width : double

+ GetLength() : double

+ SetLength(double value) : void

+ GetLength() : double

+ SetLength(double value) : void

+ GetLength() : double

+ Rectangle()

Writing code for the fields of a class

- Class fields (instance variables) are to be declared. A variables must have a type and a value. Initial values are set by the compiler.
  - All types are initiated to their "zero" values. Numeric types receive a value 0 and objects are initiated to null.
  - The keyword null means "nothing" . An object-reference has nothing to hold on, i.e. the object is not created. A null reference does not refer to any object.

```
public class Rectangle
{
        private double length;
        private double width;
}
```

Member visibility

- All identifiers including classes, constructors, fields, properties and methods of a class have accessibility that determines how accessible (visible) they are to other classes.
- An access modifier is a C# keyword that indicates how a class member, a field or method, can be accessed.
- Accessibility in C# is set by access modifiers: public, protected, private, internal and protected internal.
- When no access modifier is specified, the compiler uses default modifiers for a class and a class member.
- The modifier protected and protected internal are used with inheritance that is not a part of this basic course.
  - Derived classes also concern inheritance. A derived class is a child class of a parent class.

Access modifiers

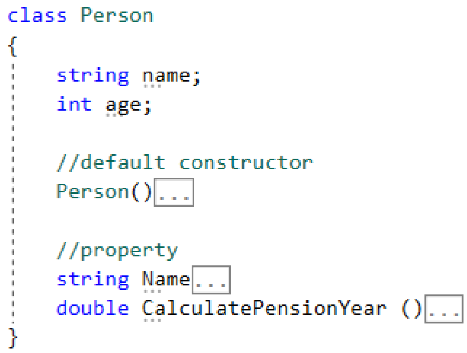
- public: The type or member can be accessed by any other code in the same assembly or another assembly that references it.
- private: The type or member can be accessed only by code in the same class.
- protected: The type or member can be accessed only by code in the same class, or in a class that is derived (inheritance) from that class.
- internal: The type or member can be accessed by any code in the same assembly, but not from another assembly.
- protected internal: The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
- private protected: The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

Default access modifiers

- A class (or a type) can be declared public or internal but not private unless it is nested inside another class.
- When no modifier is specified, the default modifier is applied,
- Default access modifiers:
  - at namespace level internal
  - at a class level private
  - at a method level private
- From the above, the following conclusion can be drawn:
  - The default access modifier for classes is internal. Availability with the same namespace.
  - Fields – private. Availability within the same class.
  - Methods – private. Availability within the same class.

Default modifiers - example

- The class Person is written without any modifiers. Default modifiers are applied.
- The class' accessibility is internal, i.e. the class **Person** is available for all classes within the same namespace.
- The fields name and age are private by default (good)!
- The constructor is private which prevents instantiation of the class – not good! It must be public.



```
class Person
{
    string name;
    int age;

    //default constructor
    Person()...

    //property
    string Name...
    double CalculatePensionYear ()...
}
```

- **Name** is a property, but it is private by default in the example – not good! It must be public because it is not needed otherwise. A Property is used to give access to a private field. It should be available
- The method **CalculatePensionYear** is private by default. If the method is used internally in the class, it is ok, else if other classes could use it, it should be public.

Encapsulation

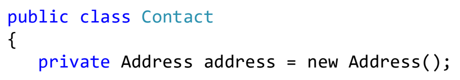
- Encapsulation is an object-oriented technique that hides the data and the internal implementation of the class.
- It means to group related data and operations in an entity which is a class in Object Orientation Programming (OOP) terms, and to control how they can be accessed from outside the class.
- A class should only contain data types that describe objects of the class and operations that are limited to the responsibility (purpose) of the class.
  - A class should not do other class' job. It should be concerned only about its own task.
- In practical term, the above is achieved by breaking a problem into smaller ones.
- Each part should then be handled by a class. Classes can collaborate in different ways to solve the big problem . Use many classes, one class for one type of objects.
  - A contact class should not handle street, city and other address items. Let the class Address take care of address items.

Encapsulation and data hiding

- Encapsulation is achieved effectively by data hiding.
  - Data hiding is done by declaring fields of a class as private.
  - Access can be provided by methods and Properties, under controlled conditions.
- Data hiding implies that data (fields) of a class should not be exposed outside the object for direct access.
- Applying encapsulation and data hiding, the data of objects are protected against accidental, unwanted, willful or un-willful modifications or use.
- Even access to the methods of a class should be limited to those which are intended for communication with other objects.
- Encapsulation of methods allows you to change the code in a class without worrying about existing code that uses the methods of the class.

Communication between objects of classes

- Object of classes need to use each other in an application in order to solve a given problem. Classes communicate in different ways:



```
public class Contact
{
    private Address address = new Address();
```

  - Aggregation: one class can use an object of another class. The other class's public members will be then available for the host (owner) class.
  - Inheritance: one class can directly inherits all public and protected members of a parent class. Inheritance is not discussed in this course.
- When a class uses an object of another class, it must create an object of the other class using the keyword new (with the exception of static and abstract classes that are not included among the topics of this course).
- The class can then access all the public members declared in the object's class.
- Every object that is created will have its own set of values in the object's fields and its own copy of the methods.

Member accessibility - recommendations

- Fields of a class should always be declared as `private`.
- This is in accordance to one of the important principles of the object-oriented programming, namely encapsulation and data hiding.
- Only constants that are `static` can in some cases be declared as `public.`
- `public` fields should strictly be avoided. Auto-implemented properties can be used instead.
- Methods should be declared public when they provide services to other classes, i.e. when the method is a part of the interface of the class.
- Methods that are used only by other methods internally in the same class, should always be private.

Other recommendations

- Classes usually need to be `public` or `internal` so that other classes can make use of them.
- Although more than one `public` class can be placed in the same file, it is very common to have one class per file, no matter how little the class may be.
- A class name does not need to match the file name. But you should always name it the same name as the class when possible.
- The class structure, organized in namespaces and directories, need not to map to same file structure on the computer.
- When methods need to be called by other objects in a sequence, it may be practical to make them private and encapsulate them inside a public method
- This public method serves then as a part of the class's interface towards other classes.

```
public void Calculate()
{
    ReadInput();     //private method
    Compute();       //private method
    ShowResults();   //private method
}
```

Summary

- A class is a definition for a group of objects and is used create instances of.
- Identify many object groups in a problem and write a class for each group. Let each class have a well-defined area of responsibility.
- Each instance describes a particular object that is saved in the computer's memory.
- Encapsulation is about protecting data and encapsulating implementations in a class.
- A class cannot be private or protected but can be public or internal. The default accessibility is internal, which makes the class accessible for other classes in the same namespace.
- Fields and methods of class have private as the default modifier.
- All instance variables (fields) are to be declared as private.
- Constructors should always be declared public. Otherwise, the class cannot be instantiated.
- Methods that are not intended to be used by other classes should be private. If you are not sure about it, let them be public.
- Think always object-oriented, without regard to the size of the application.