**Tutorial C# Basics 8**

**OOP 1/2 – the basics**

V1.0

11/08/2022

In this tutorial and the next, you will study THE most important programming pattern – Object-Oriented Programming (or OOP). We will cover the very basics of OOP in this tutorial and explore the in-depth concepts in the next one. By the end of this tutorial, you will be able to

* Understand the OOP basic concepts
* Define a class with member fields, member methods, and constructors
* Create objects of a class
* Tell the difference between static and non-static members.

**Topics**

[**1.** **The concept of OOP** 1](#_Toc111128512)

[**2.** **Classes and objects** 2](#_Toc111128513)

[**Step 1.** **Create the Car class** 2](#_Toc111128514)

[**Step 2.** **Implement the Car class** 4](#_Toc111128515)

[**Step 3.** **Create an object and access the field “color”** 4](#_Toc111128516)

[**3.** **Class members & access modifiers** 5](#_Toc111128517)

[**4.** **Constructors** 5](#_Toc111128518)

[**Class activity 1** 6](#_Toc111128519)

[**5.** **A todo program** 6](#_Toc111128520)

[**Step 1.** **Define the todo item class** 7](#_Toc111128521)

[**Step 2.** **Test the TodoItem class** 7](#_Toc111128522)

[**Step 3.** **Define the todo list manager class** 8](#_Toc111128523)

[**Step 4.** **Test the TodoManager class** 9](#_Toc111128524)

[**Step 5.** **Improve your program** 10](#_Toc111128525)

[**6.** **Static and none-static members** 11](#_Toc111128526)

[**Model answers** 11](#_Toc111128527)

[**Class activity 1** 11](#_Toc111128528)

1. **The concept of OOP**

Study the [C# OOP](https://www.w3schools.com/cs/cs_oop.php) tutorial on w3schools to understand the basic Object-Oriented Programming concepts. There are two important terms mentioned in the tutorial – class and object.

In computer memory, class doesn't occupy memory, but it imparts an idea about the features of objects. Objects, on the other hand, are the instances of the class. An object is an actual entity that occupies space in the memory.

Think about **class** is blueprint – the design of building on paper, and **objects** are the actual buildings constructed using the blueprint.



1. **Classes and objects**

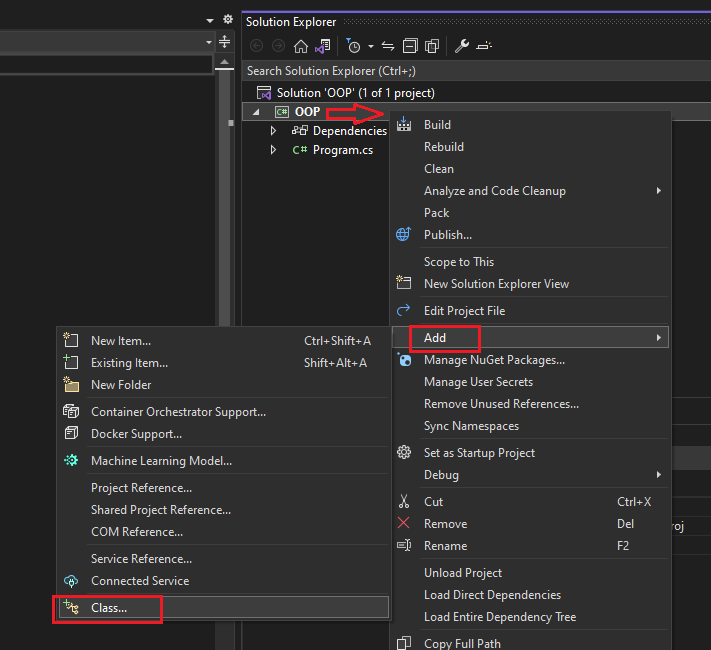
Study the [C# classes/objects](https://www.w3schools.com/cs/cs_classes.php) tutorial on w3schools. Takeaway notes from the tutorial

* In OOP, we always start with creating a class and then creating objects using the class.
* **Fields** can be defined directly in a class. Fields are “attributes” of the class, e.g. color / make of a car.
* Operator **new** is used to create an object of a class. Essentially, the operator **new** will allocate a block of memory from the system memory pool for the object.
* For now, always put the key word **public** before the word class and before each field.
  + We will talk about the key word “public” later in this tutorial.

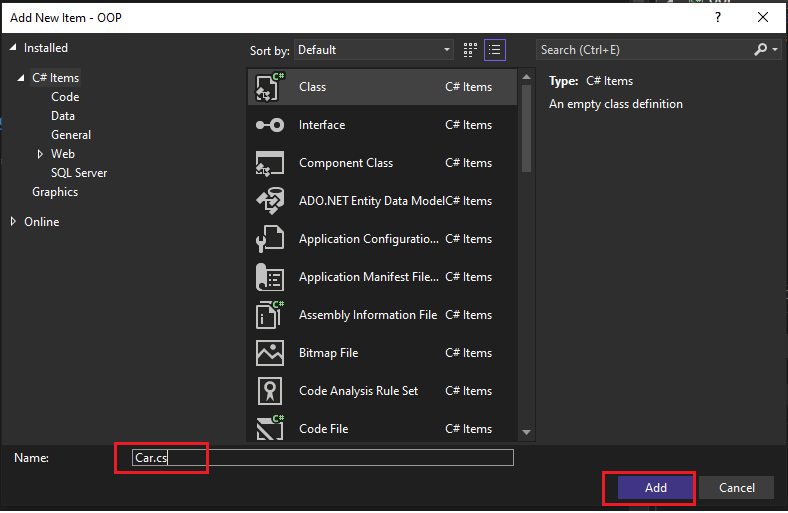
Put together all that’s been mentioned in this tutorial, and let’s do it again in Visual Studio.

* 1. **Create the Car class**

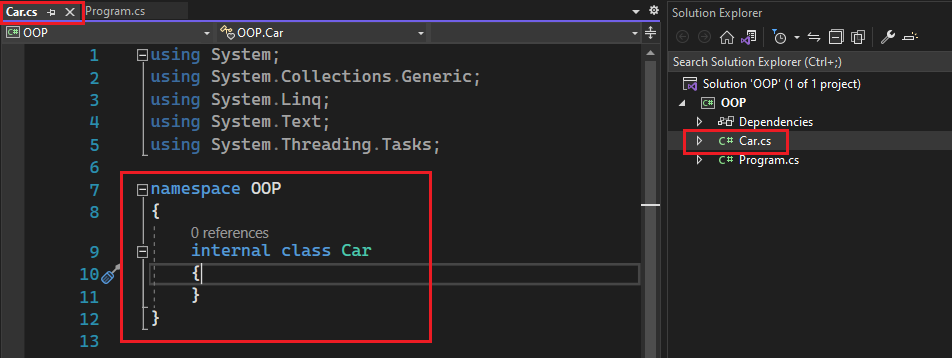
In Visual Studio, create a project and from the “solution explore” window 🡪 right click on the project name 🡪 from the pop-up menu, select “Add” 🡪 select “class”



In the “Add new item” window, set the file name “Car.cs” 🡪 click on Add button.

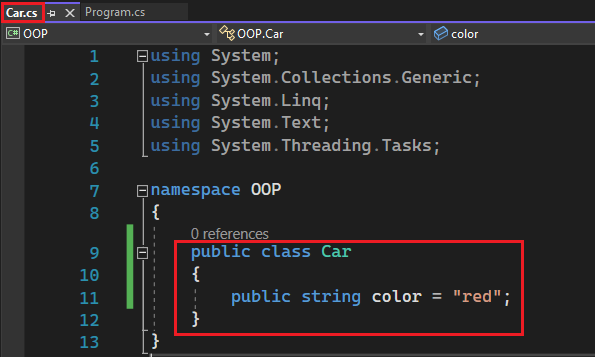


A new file called “Car.cs” is then created, and you can see it in the Solution explorer window. Open the file in the code editor to view its contents.



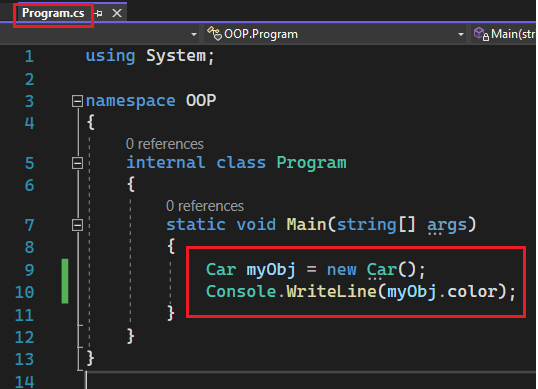
* 1. **Implement the Car class**

Now, in the code editor, implement the Car class as below. Make sure you added the key word “**public**” before the word “class” and another “**public**” before the variable color.

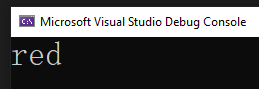


* 1. **Create an object and access the field “color”**

In the Program.cs, create a car object and display the color information.



Run the program and check the result



1. **Class members & access modifiers**

Study the [C# Class Members](https://www.w3schools.com/cs/cs_class_members.php) tutorial on the w3schools website. Takeaway notes:

* There are two type of class members – field and method
  + Fields are variables used to hold data. Also with fields, a class is essentially a custom complex data type
  + Methods define the behaviours of the class.

Study the [C# Access Modifier](https://www.w3schools.com/cs/cs_access_modifiers.php) tutorial on the w3schools website. Takeaway notes:

Graphical user interface, text, application, email

Description automatically generated

Practice with the examples used in the tutorials in Visual Studio.

1. **Constructors**

Study the [C# Constructors](https://www.w3schools.com/cs/cs_constructors.php) tutorial on the w3schools website. Takeaway notes:

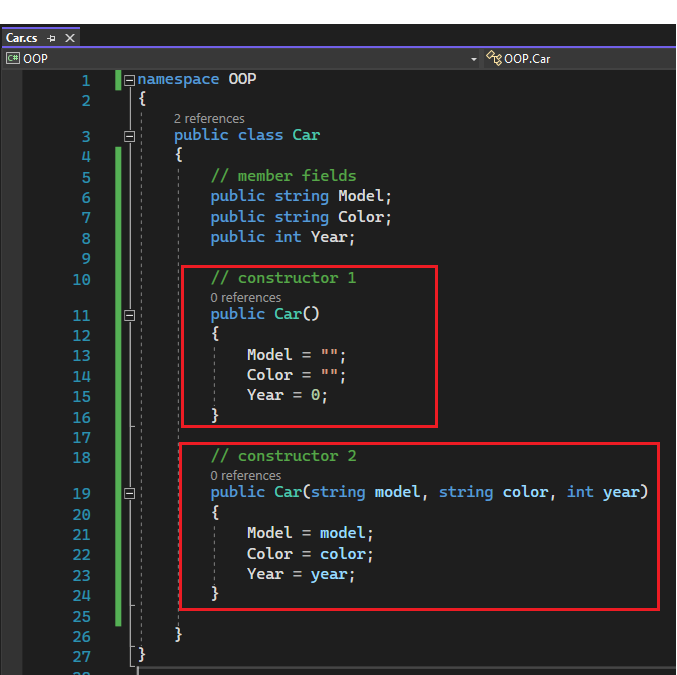
* Constructor name must match the class name, and it cannot have a return type
* Constructor is a method that will be automatically called upon creating the object.
* Usually, constructor is used to initialize member fields.

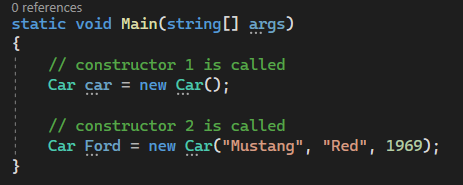
Practice with the example from the w3schools C# Constructors tutorial in Visual Studio.

Study the concept of [C# Method Overloading](https://www.w3schools.com/cs/cs_method_overloading.php), and let’s create 2 version of constructors for a class

* The first constructor has no parameter, and all member fields are initialized with a default value
* The second constructor has parameters, and each of the parameters is used to set the value of a member field

Let’s use the Car class example used in the w3schools tutorial and implement two versions of constructor for the class. Here are the codes





It is a good practice to have multiple versions of constructor for a class, to give users more options when creating an object.

**Class activity 1**

Use the Car class developed above, create a list of cars using C# List. Then display the whole list in console. Use the C# List feature studied in the previous tutorial.

|  |
| --- |
| Your codes: |
| Test results: |

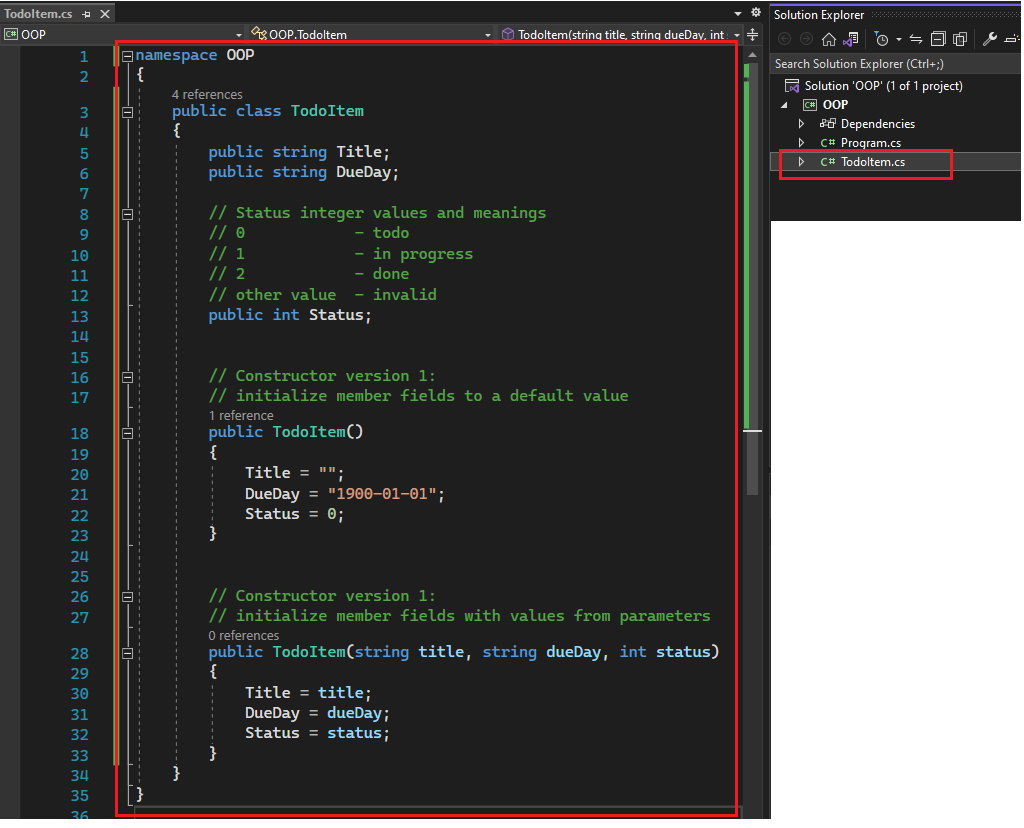
1. **A todo program**

This program manages a todo list for a single user. User could create / update / delete any todo items in the list. A todo item has the following attributes

* Title
* Due day
* Status (todo, in progress, done)
  1. **Define the todo item class**

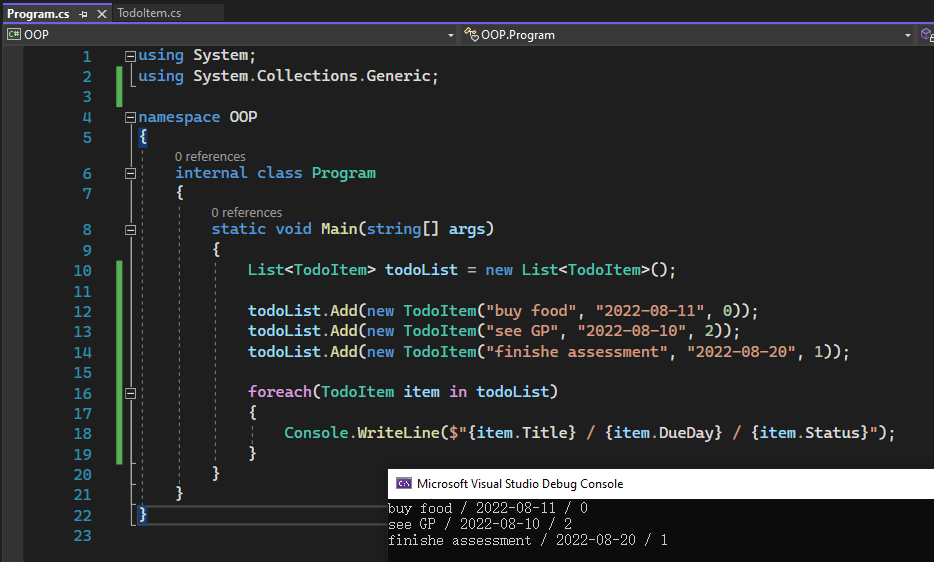
Let’s start with defining the todo item class. Create a new console project and create a new .cs file called TodoItem.cs, with the following contents. Please read the inline comments to understand the codes.

**Note**: please change the class “public” from “internal”.



* 1. **Test the TodoItem class**

Before moving on, let’s write some codes in the main program for testing purpose. Create a list of the items and display them all.



* 1. **Define the todo list manager class**

The TodoItem looks good. Let’s move on and create a class to manage the list of todo items. The manager class maintains a list of TodoItem and need to support the following operations

* Add a todo item and append it to the end of the list
* Set the status of a todo item
* Display the todo item list
* … and more …

The following code implemented the first 3 item above. You may challenge yourself to define and implement more operations, such as delete, change title/due day …

Read the comments inline to understand the codes.

|  |
| --- |
| using System;  using System.Collections.Generic;  namespace OOP  {  public class TodoManager  {  // To achieve "Encapsulation", let's make the todo list private.  // So that users have to call the member methods to operate the todo items.  private List<TodoItem> TodoList;  // This is the constructor, in which the TodoList is initialized.  public TodoManager()  {  TodoList = new List<TodoItem>();  }  // Add: create a new todo item and append it to the end of the todo list  public void Add()  {  // take user input for title and due day  Console.WriteLine("Creating a new todo item:");  Console.Write("Enter the title > ");  string title = Console.ReadLine();  Console.Write("Enter the due day (yyyy-mm-dd) > ");  string dueDay = Console.ReadLine();    // create a new item to the end of the todo list with a initial status 0 (todo)  TodoList.Add(new TodoItem(title, dueDay, 0));  }  // SetStatus: set the status of item by index  public void SetStatus()  {  // take user input for title and due day  Console.WriteLine("Set the status of an item:");  Console.Write("Enter the index > ");  int index = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter the new status (0 - todo, 1 - in progress, 2 - done) > ");  int newStatus = Convert.ToInt32(Console.ReadLine());  if (index < 0 || index >= TodoList.Count)  {  Console.WriteLine("index entered does not exist.");  return;  }  // update the status  TodoList[index].Status = newStatus;  // display the updated item  Console.WriteLine("Status updated for item");  Console.WriteLine($"Index: {index}");  Console.WriteLine($"Title: {TodoList[index].Title}");  Console.WriteLine($"Due day: {TodoList[index].DueDay}");  Console.WriteLine($"Status: {TodoList[index].Status}");  }  // Display: print the whole list to the console  public void Display()  {  Console.WriteLine("The todo list");  foreach (TodoItem item in TodoList)  {  Console.WriteLine($"{TodoList.IndexOf(item)} | {item.Title} | {item.DueDay} | {item.Status}");  }  }  }  } |

* 1. **Test the TodoManager class**

write some codes in the main program for testing purpose, like so

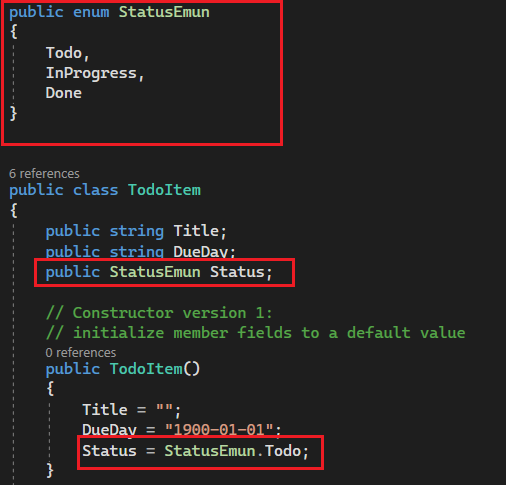
Text

Description automatically generated

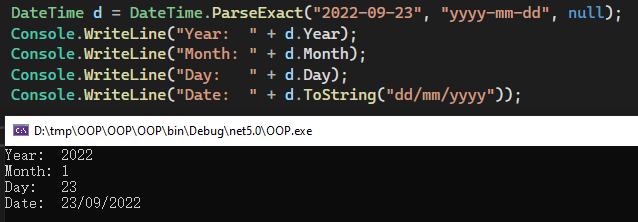
* 1. **Improve your program**

Challenge yourself to improve the program in different ways. Here are some examples

* Incorporate the command line interface structure into the program to make it a real application
* Define and implement more operations for the program, such as delete item, change title/due day
* Study C# Enums feature and convert the item status to an emun type. Here is the code sample



* Convert the type of DueDay to C# DateTime. And here is the code sample



1. **Static and none-static members**

It is the time to explain the “static” key word you have used a lot before. Use the following tutorial to study the concept of the static field, static method, and static class

<https://www.tutorialsteacher.com/csharp/csharp-static>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members>

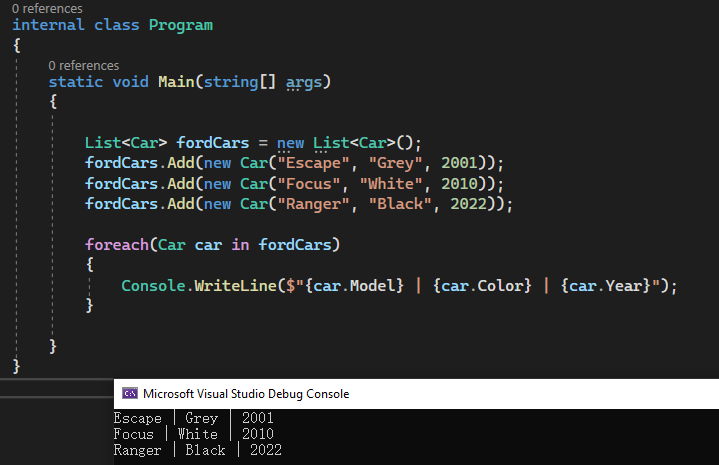
The IO methods we used from the beginning in C#, such as the

* Console.WriteLine
* Console.ReadLine

are all static methods. To use these methods, we do not have to create a console instance to call the methods.

**Model answers**

**Class activity 1**



End of this tutorial