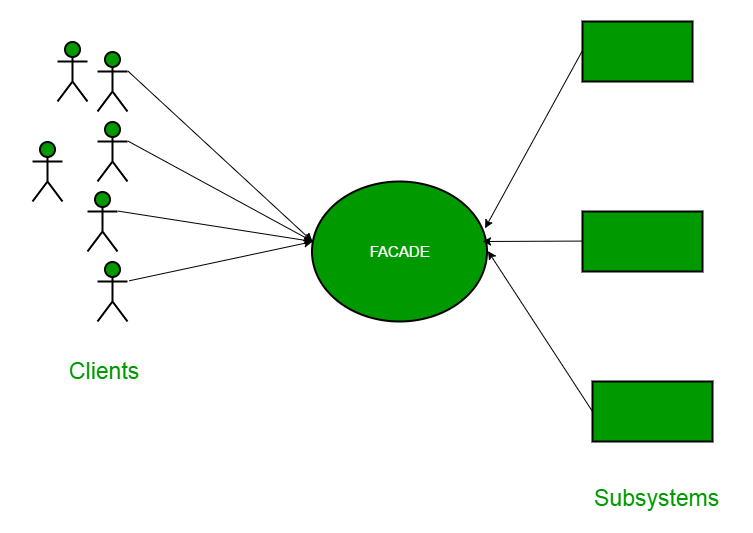
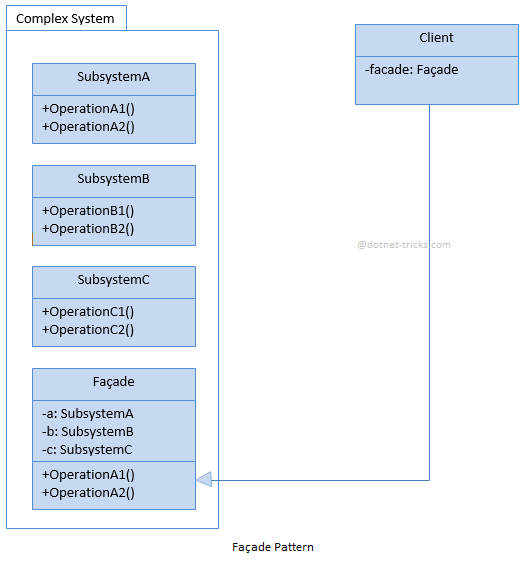
**Facade Design Pattern**.

It hides the complexities of the system and provides an interface to the client from where the client can access the system.

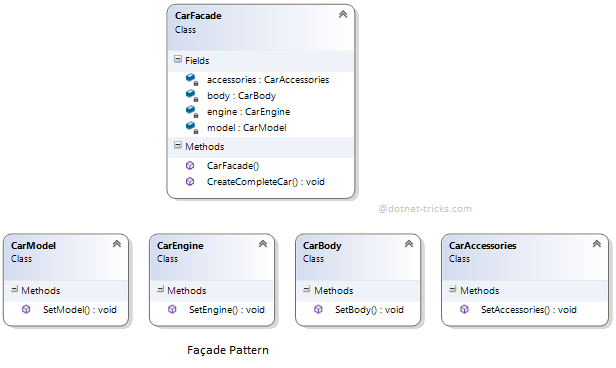
**Example** : Let’s consider a hotel. This hotel has a hotel keeper. There are a lot of restaurants inside hotel e.g. Veg restaurants, Non-Veg restaurants and Veg/Non Both restaurants.  
You, as client want access to different menus of different restaurants . You do not know what are the different menus they have. You just have access to hotel keeper who knows his hotel well. Whichever menu you want, you tell the hotel keeper and he takes it out of from the respective restaurants and hands it over to you. Here, the hotel keeper acts as the **facade**, as he hides the complexities of the system hotel.



**UML Class Diagram**



**Example**



Who is what?

The classes, interfaces, and objects in the above class diagram can be identified as follows:

1. **CarModel, CarEngine, CarBody, CarAccessories** - These are subsystems.
2. **CarFacade**- Facade class.

C# - Sample Code

/// <summary>

/// The 'Subsystem ClassA' class

/// </summary>

class CarModel

{

public void SetModel()

{

Console.WriteLine(" CarModel - SetModel");

}

}

/// <summary>

/// The 'Subsystem ClassB' class

/// </summary>

class CarEngine

{

public void SetEngine()

{

Console.WriteLine(" CarEngine - SetEngine");

}

}

/// <summary>

/// The 'Subsystem ClassC' class

/// </summary>

class CarBody

{

public void SetBody()

{

Console.WriteLine(" CarBody - SetBody");

}

}

/// <summary>

/// The 'Subsystem ClassD' class

/// </summary>

class CarAccessories

{

public void SetAccessories()

{

Console.WriteLine(" CarAccessories - SetAccessories");

}

}

/// <summary>

/// The 'Facade' class

/// </summary>

public class CarFacade

{

CarModel model;

CarEngine engine;

CarBody body;

CarAccessories accessories;

public CarFacade()

{

model = new CarModel();

engine = new CarEngine();

body = new CarBody();

accessories = new CarAccessories();

}

public void CreateCompleteCar()

{

Console.WriteLine("\*\*\*\*\*\*\*\* Creating a Car \*\*\*\*\*\*\*\*\*\*\n");

model.SetModel();

engine.SetEngine();

body.SetBody();

accessories.SetAccessories();

Console.WriteLine("\n\*\*\*\*\*\*\*\* Car creation complete \*\*\*\*\*\*\*\*\*\*");

}

}

/// <summary>

/// Facade Pattern Demo

/// </summary>

class Program

{

static void Main(string[] args)

{

CarFacade facade = new CarFacade();

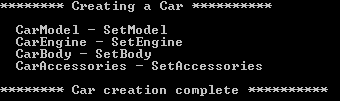
facade.CreateCompleteCar();

Console.ReadKey();

}

}

facade Pattern Demo - Output



When to use it?

1. A simple interface is required to access to a complex system.
2. The abstractions and implementations of a subsystem are tightly coupled.
3. Need an entry point to each level of layered software.
4. The facade design pattern is particularly used when a system is very complex or difficult to understand because the system has a large number of interdependent classes or its source code is unavailable